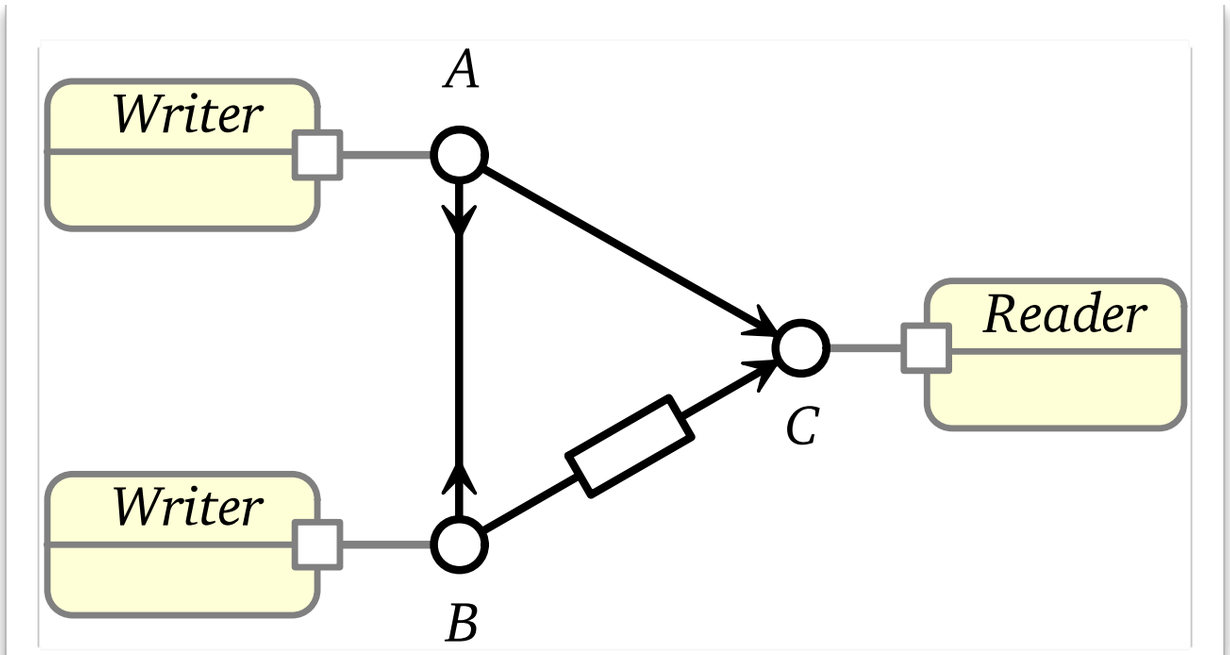
Runtime Reo

José Proença
KU Leuven, Belgium

Recall Reo



- Exogenous coordination
- Coordination is constrained interaction
- Channel ends, ports, nodes – source/sink/mixed
- Discrete atomic steps

Reo semantics

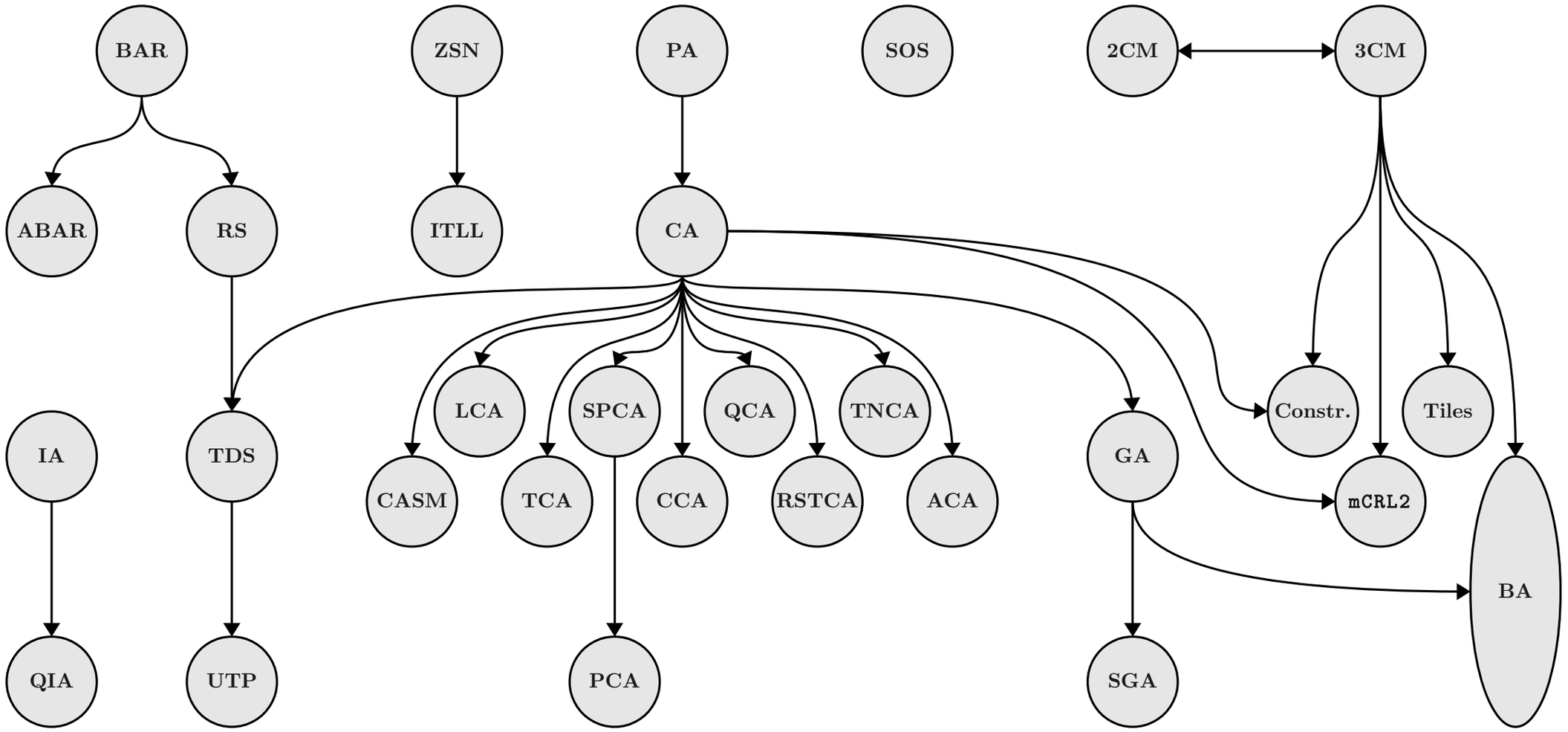
Jongmans and Arbab 2012

Overview of Thirty Semantic Formalisms for Reo

Reo semantics

- *Coalgebraic models*
  - **Timed data streams**
  - Record streams
- *Coloring models*
  - **Two colors**
  - **Three colors**
  - Tile models
- *Other models*
  - **Process algebra**
  - **Constraints**
  - Petri nets & intuitionistic logic
  - Unifying theories of programming
  - Structural operational semantics

- *Operational models*
  - **Constraint automata**
  - Variants of constraint automata
    - Timed Probabilistic
    - Continuous-time
    - Quantitative
    - Resource-sensitive timed
    - Transactional
  - Context-sensitive automata
    - Büchi automata
    - **Reo automata**
    - Intentional automata
    - Action constraint automata
    - Behavioral automata
  - Structural operational semantics



2CM : Coloring models with two colors [28, 29, 33]
3CM : Coloring models with three colors [28, 29, 33]
ABAR : Augmented BAR [39, 40]
ACA : Action CA [46]
BA : Behavioral automata [61]
BAR : Büchi automata of records [38, 40]
CA : Constraint automata [10, 17]
CASM : CA with state memory [60]
CCA : Continuous-time CA [18]
Constr. : Propositional constraints [30, 31, 32]
GA : Guarded automata [20, 21]
IA : Intentional automata [33]
ITLL : Intuitionistic temporal linear logic [27]
LCA : Labeled CA [44]
mCRL2 : Process algebra [47, 48, 49]
PA : Port automata [45]
PCA : Probabilistic CA [15]
QCA : Quantitative CA [12, 53]
QIA : Quantitative IA [13]
RS : Record streams [38, 40]
RSTCA : Resource-sensitive timed CA [51]
SGA : Stochastic GA [56, 57]
SOS : Structural operational semantics [58]
SPCA : Simple PCA [15]
TCA : Timed CA [8, 9]
TDS : Timed data streams [4, 5, 14, 62]
Tiles : Tile models [11]
TNCA : Transactional CA [54]
UTP : Unifying theories of programming [55, 52]
ZSN : Zero-safe nets [27]

Shifting to Runtime

Automata

- Good for behavioural analysis
- Good for verification
- Everything known *a priori*
- Does not scale

At runtime

- Only need the next step
- Only need 1 possible step
- Change is expected
- May scale up
  - restrict to local regions

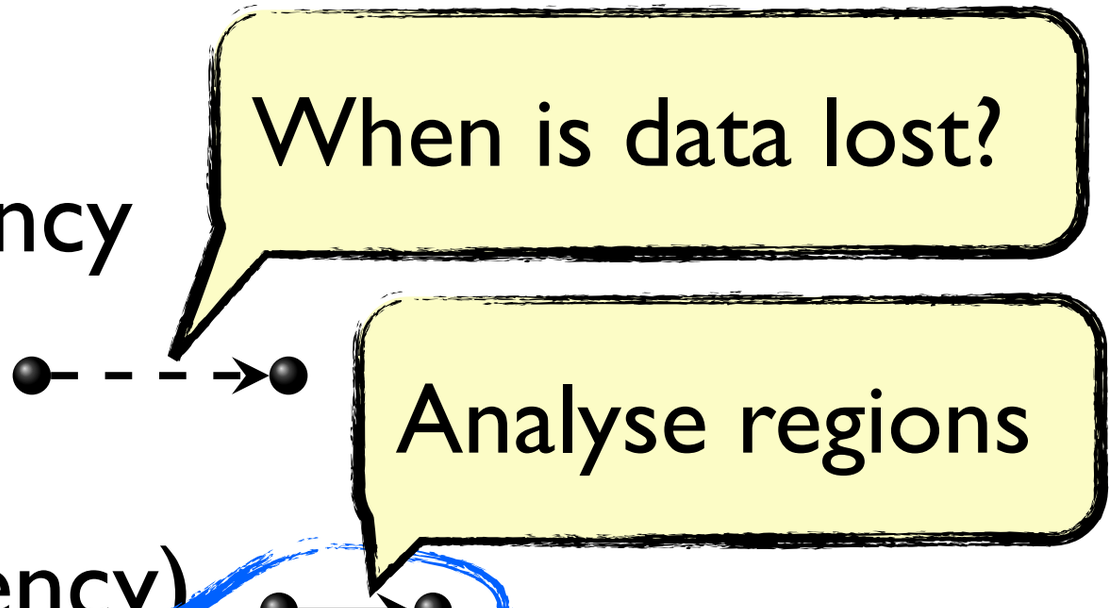
Outline

1. Context dependency
   - connector colouring 3 (CC3)[1]
2. Locality (concurrency)
   - partial connector colouring (PCC)[2]
3. Constraints
   - SAT solving with data for Reo[3]

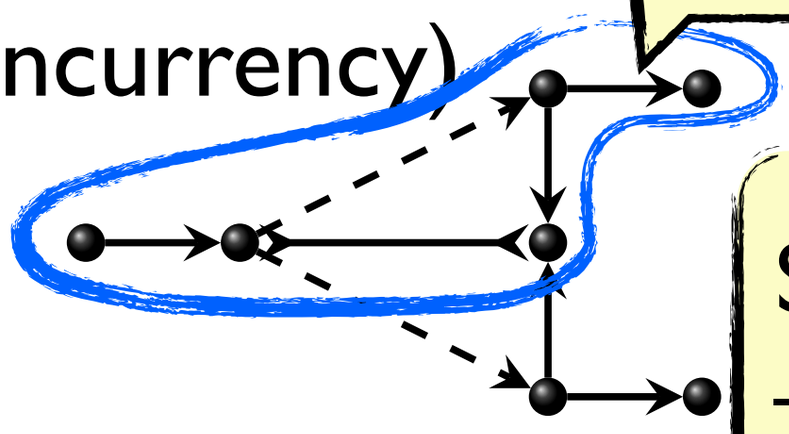
[1] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency
[2] Dave Clarke and José Proença. Partial connector colouring
[3] Dave Clarke, José Proença, Alexander Lazovik, and Farhad Arbab, Channel-based coordination via constraint satisfaction
José Proença, Dave Clarke, Interactive interaction constraints

Challenges

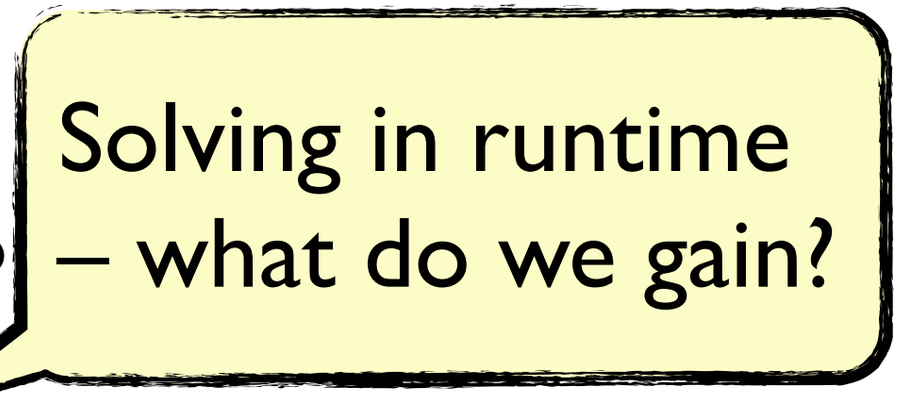
1. Context dependency



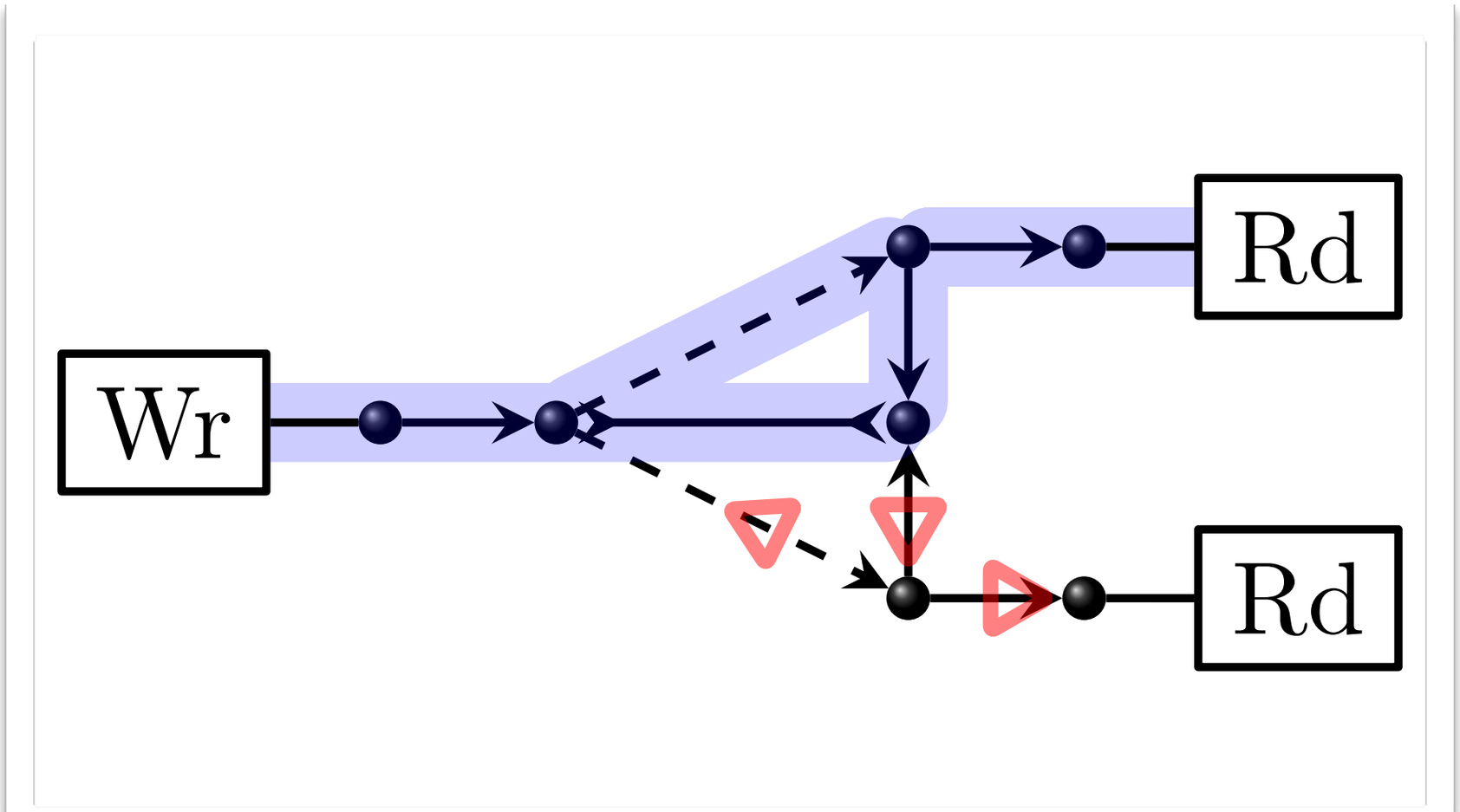
2. Locality (concurrency)



3. Constraints

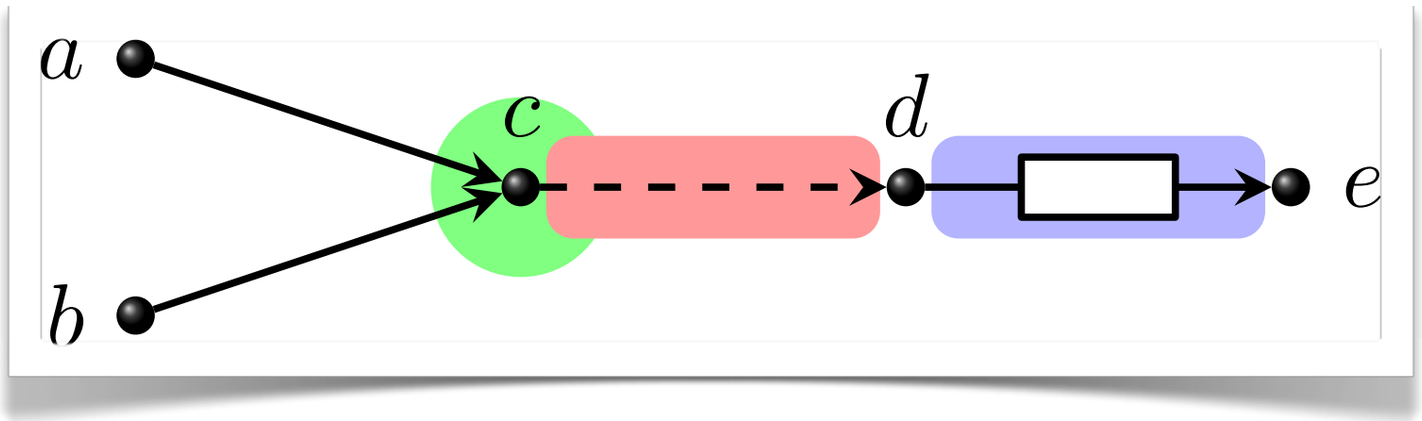


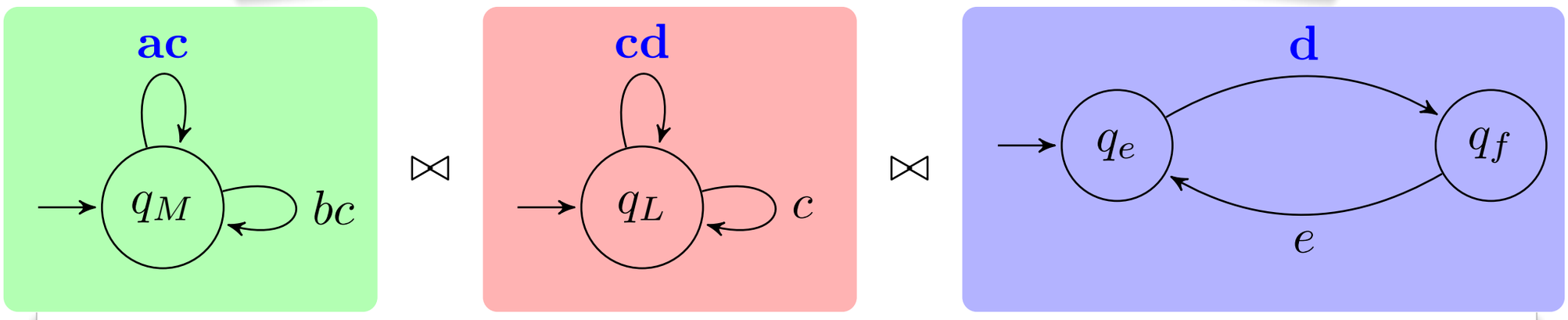
$$\text{step} \models (a \wedge b) \vee (a \wedge c) \wedge \phi$$

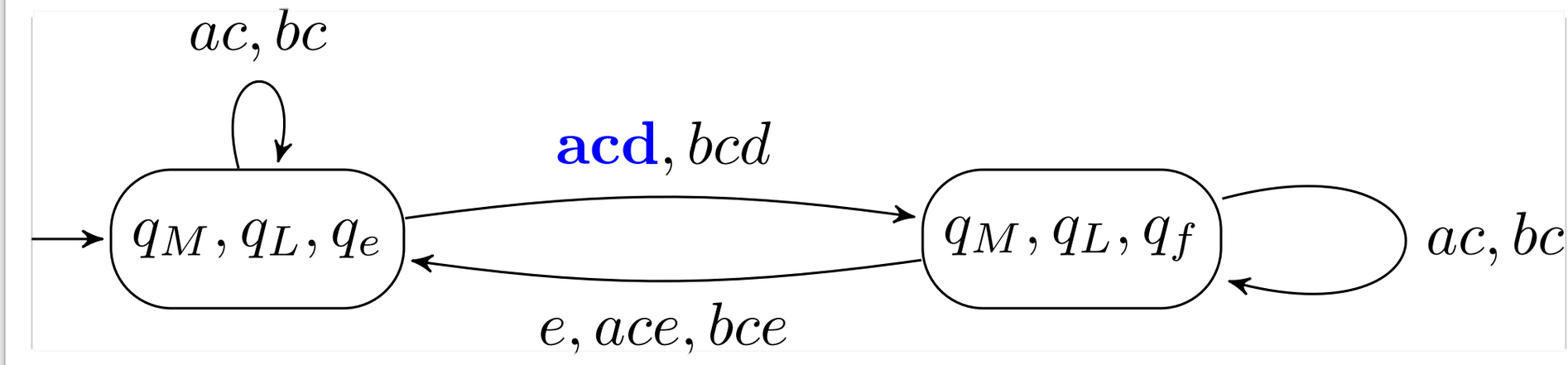


Context dependency

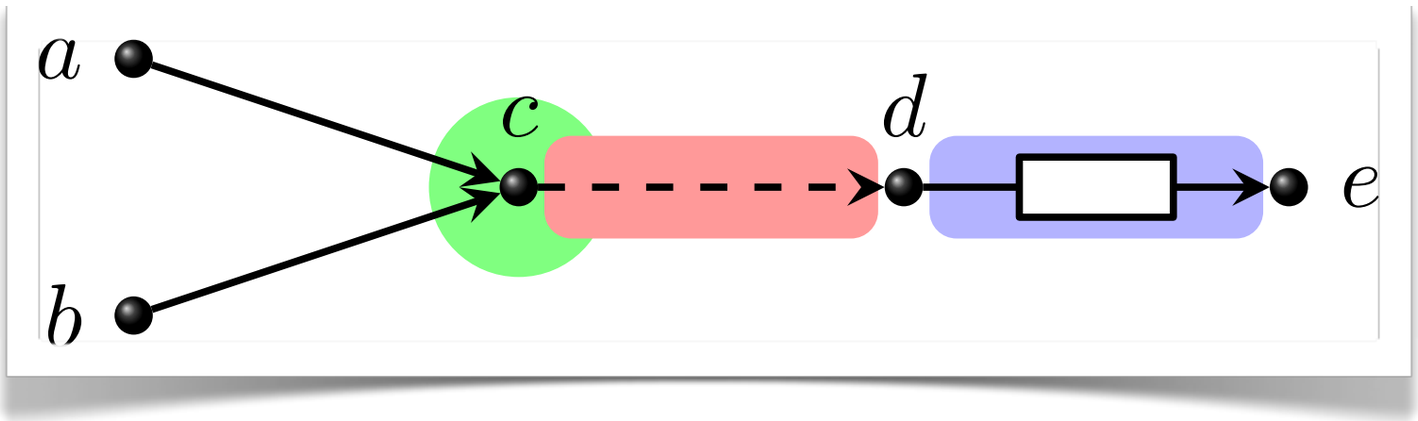
Composing steps

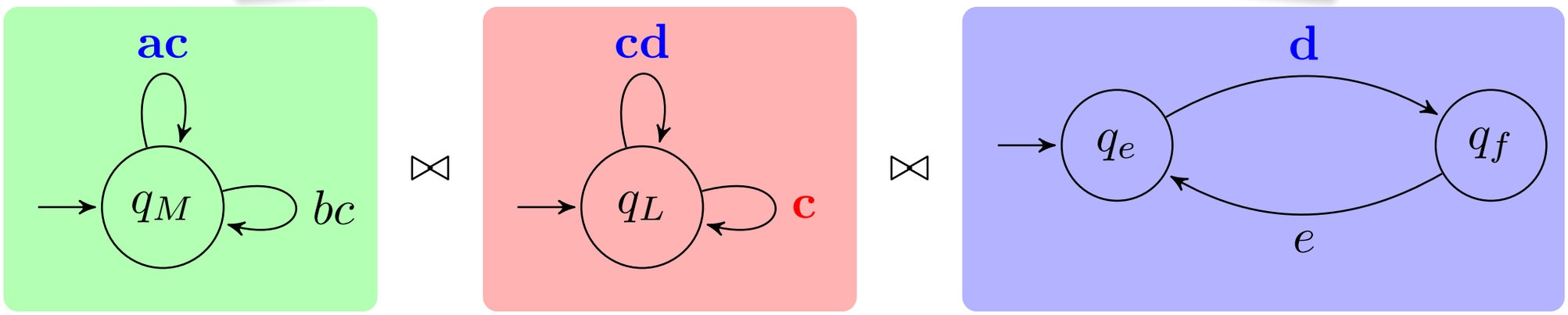






Composing steps

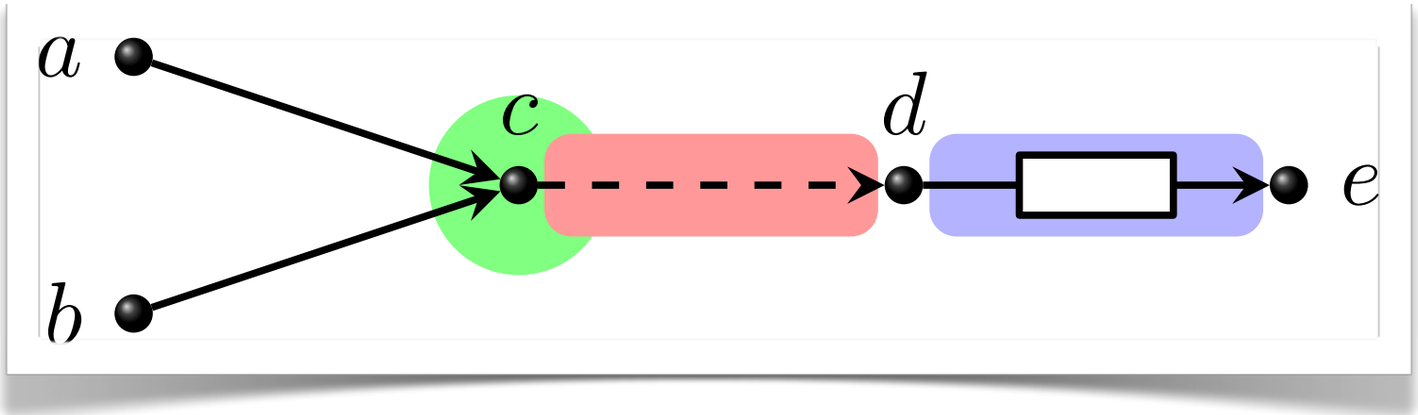


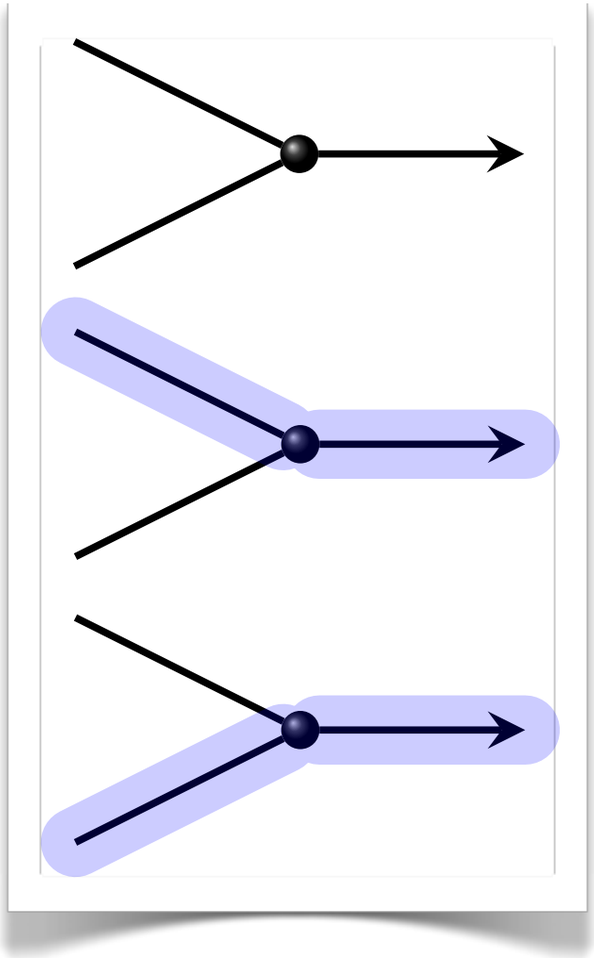


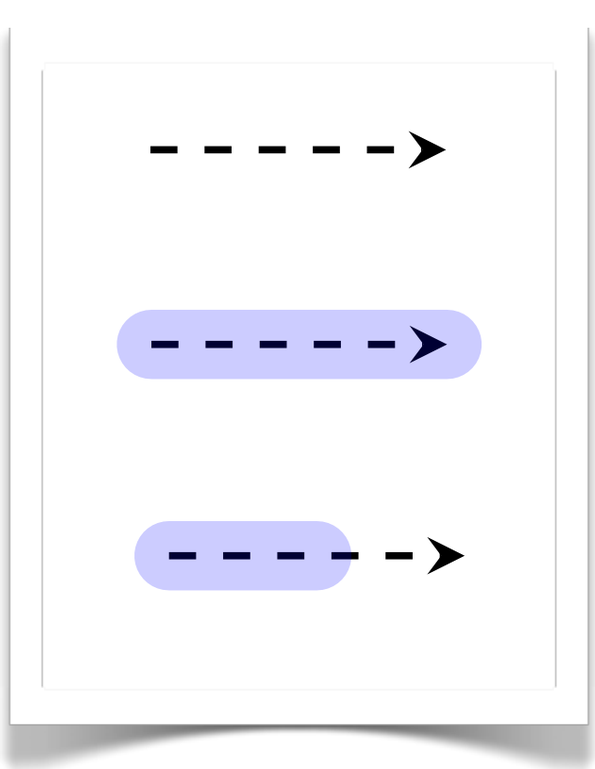
$$\mathbf{ac} \bowtie \mathbf{cd} \bowtie \mathbf{d} = \mathbf{acd}$$

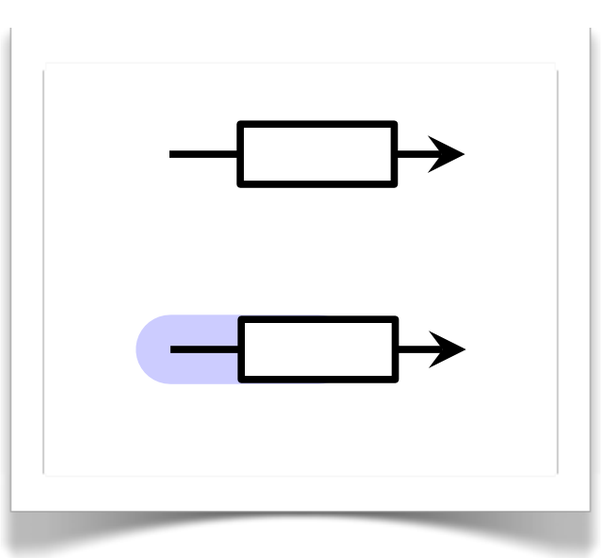
$$\mathbf{ac} \bowtie \mathbf{c} \bowtie \mathbf{d} = \bot$$

Colourings



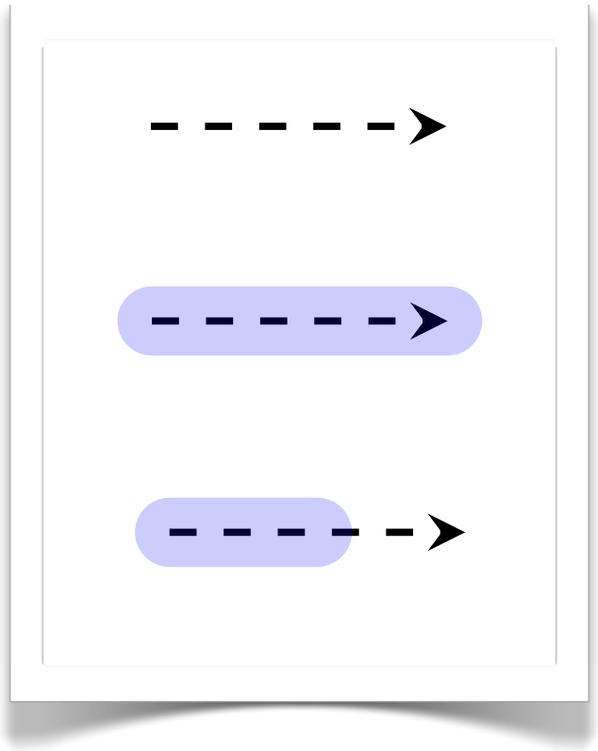






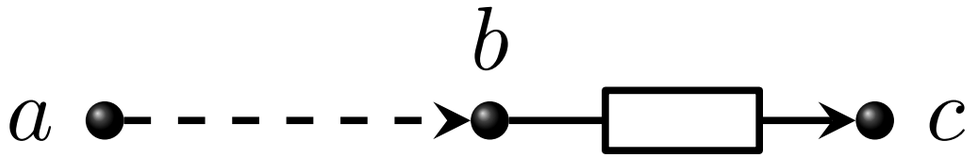
Colouring semantics

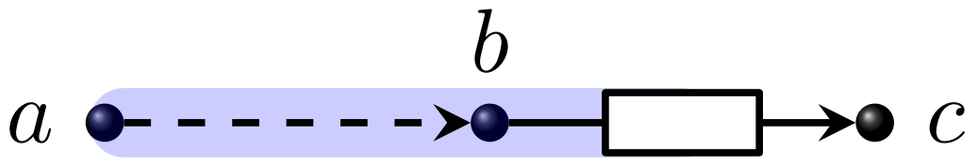
- *Colouring*: End → {Flow, NoFlow}
- *Colouring table*: Set(Colouring)
- *Composition* = matching colours

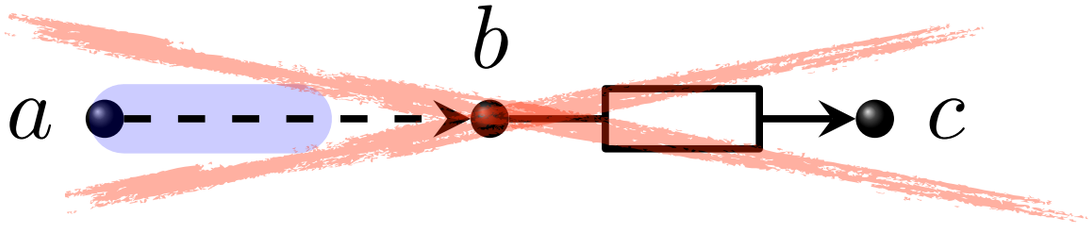


- More visual (intuitive)
- Used for generating animations

2 reasons for context

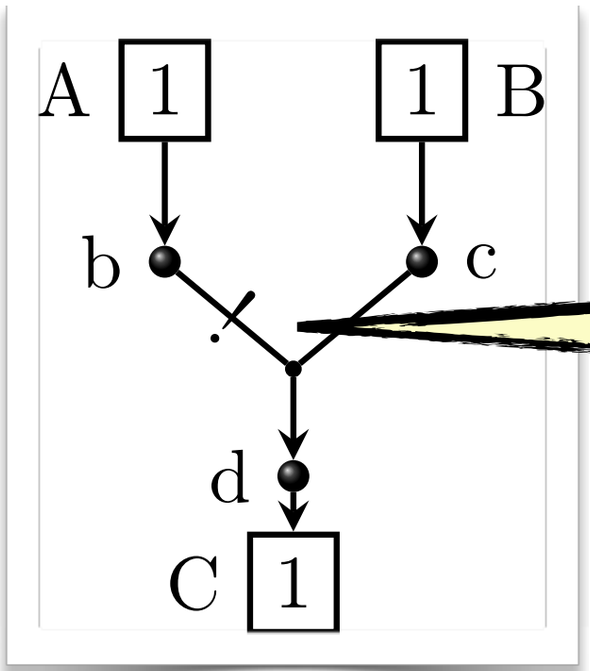




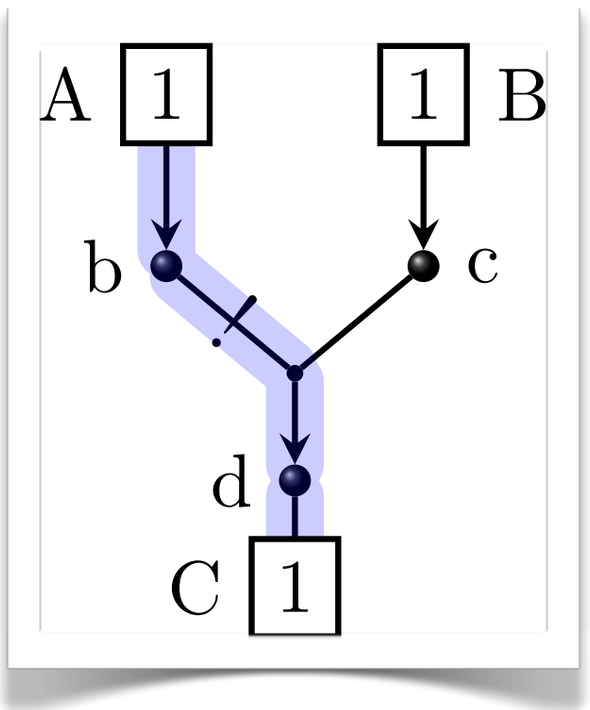


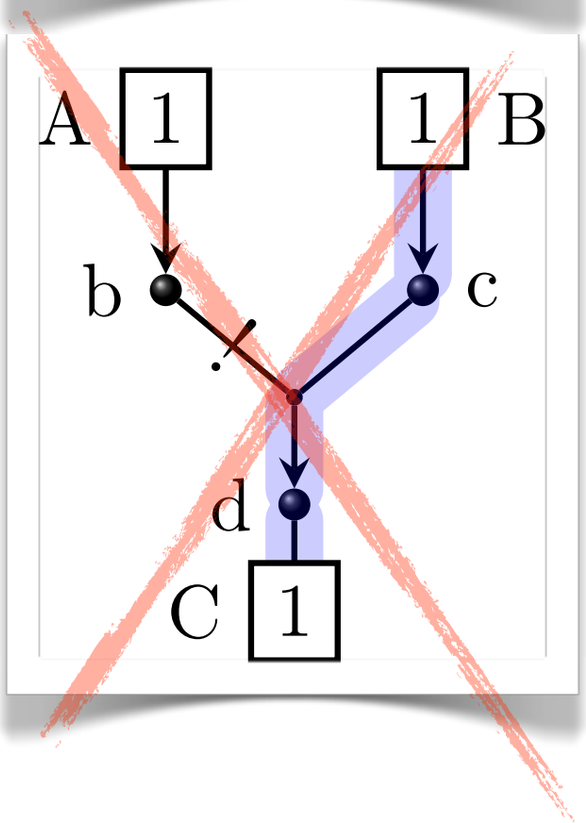
1 - avoid data loss when the context (FIFO) can receive the data.

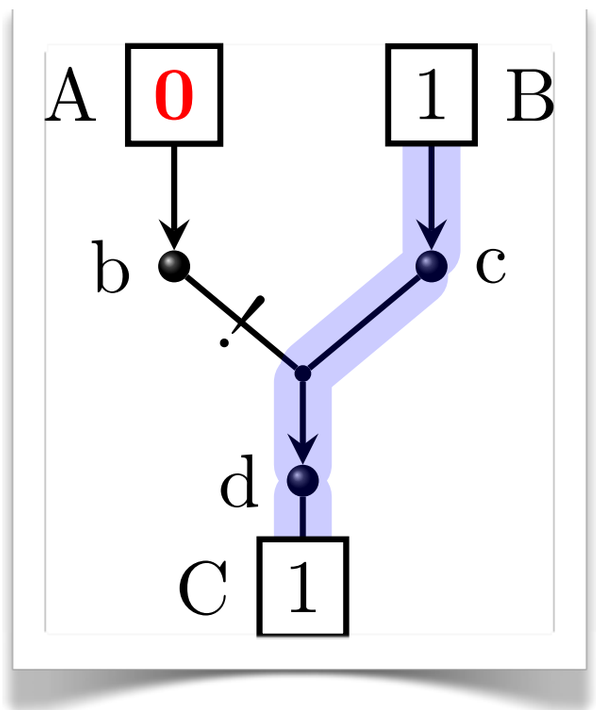
2 reasons for context



2 - give priority based on the context (writer)



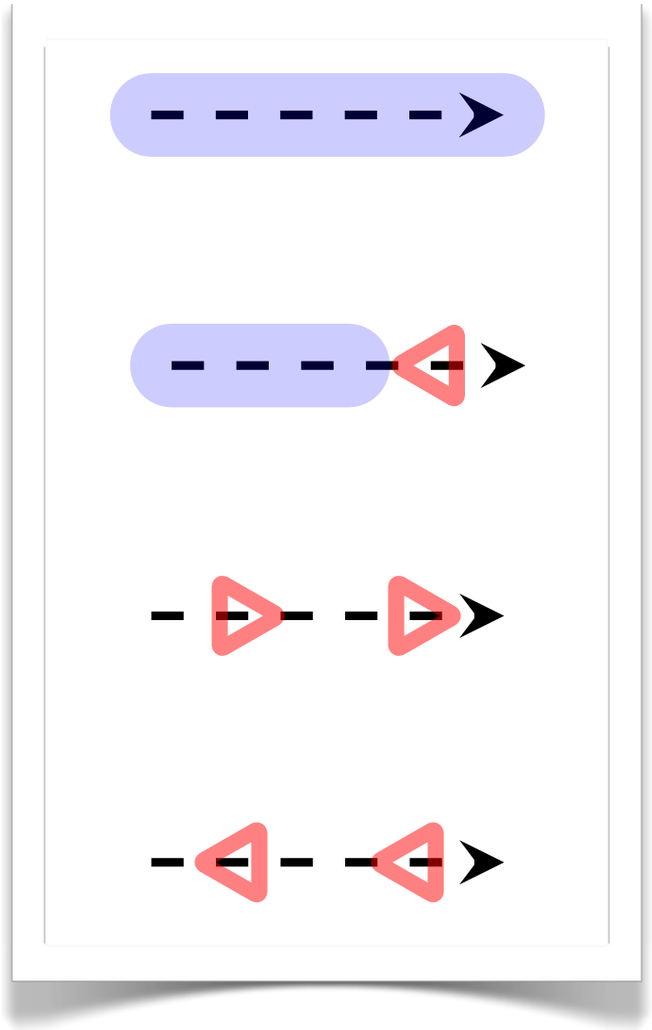




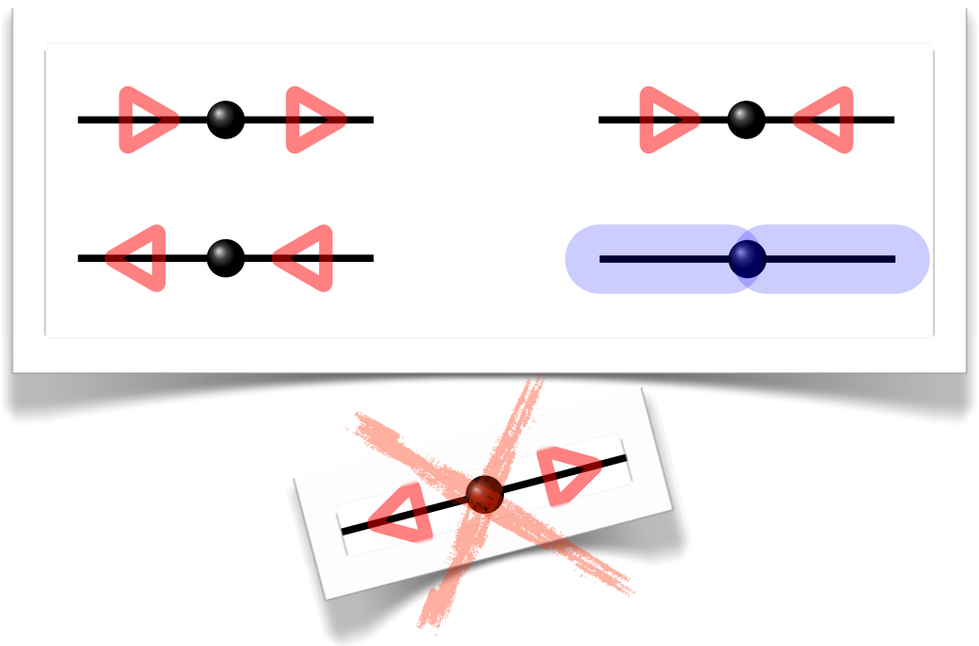
Context = 3 colours

- *Colouring*:

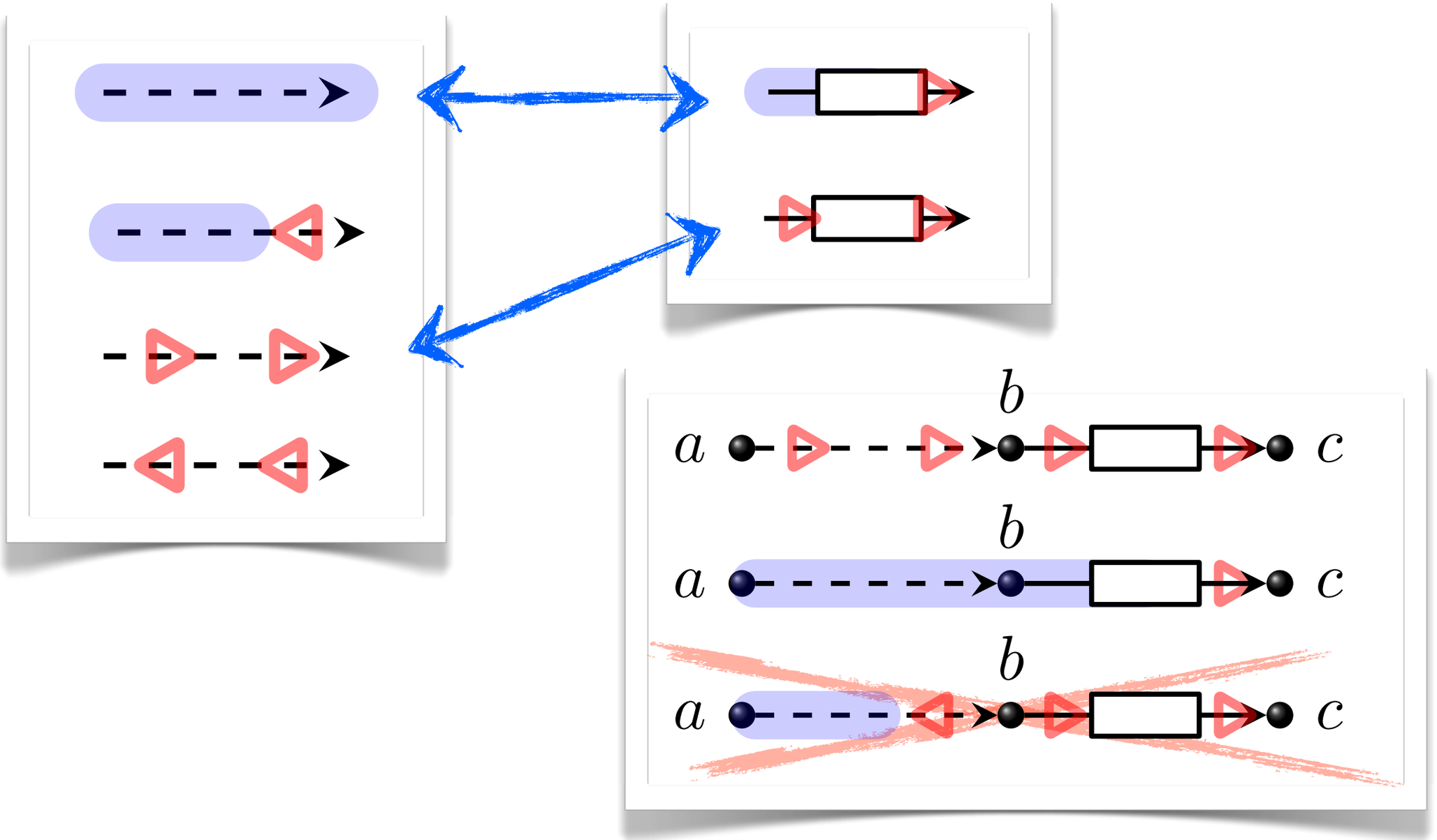
  End → {Flow, GiveReason, GetReason}



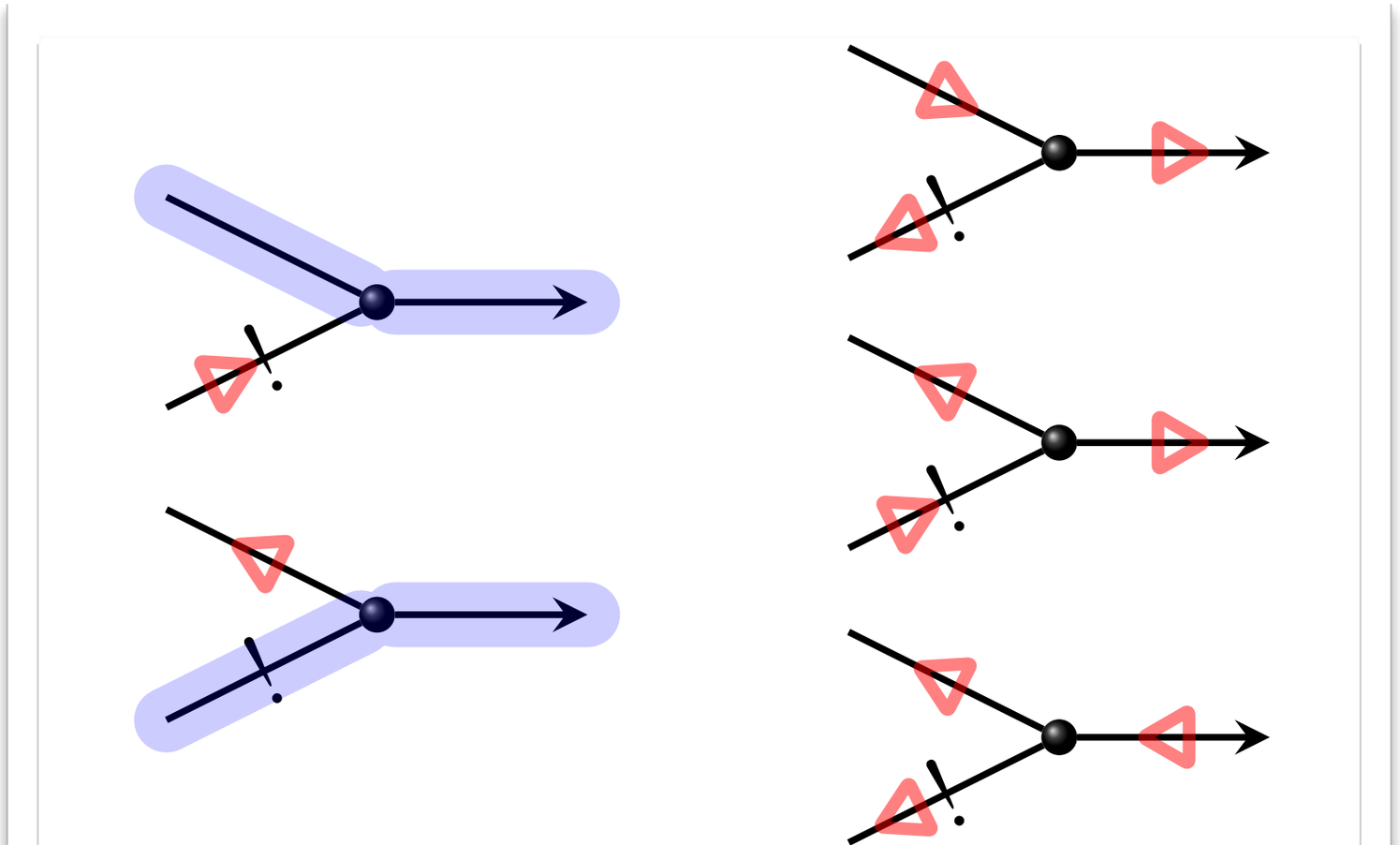
- *Composition* = matching colours:



Composition



Priority with 3 colours



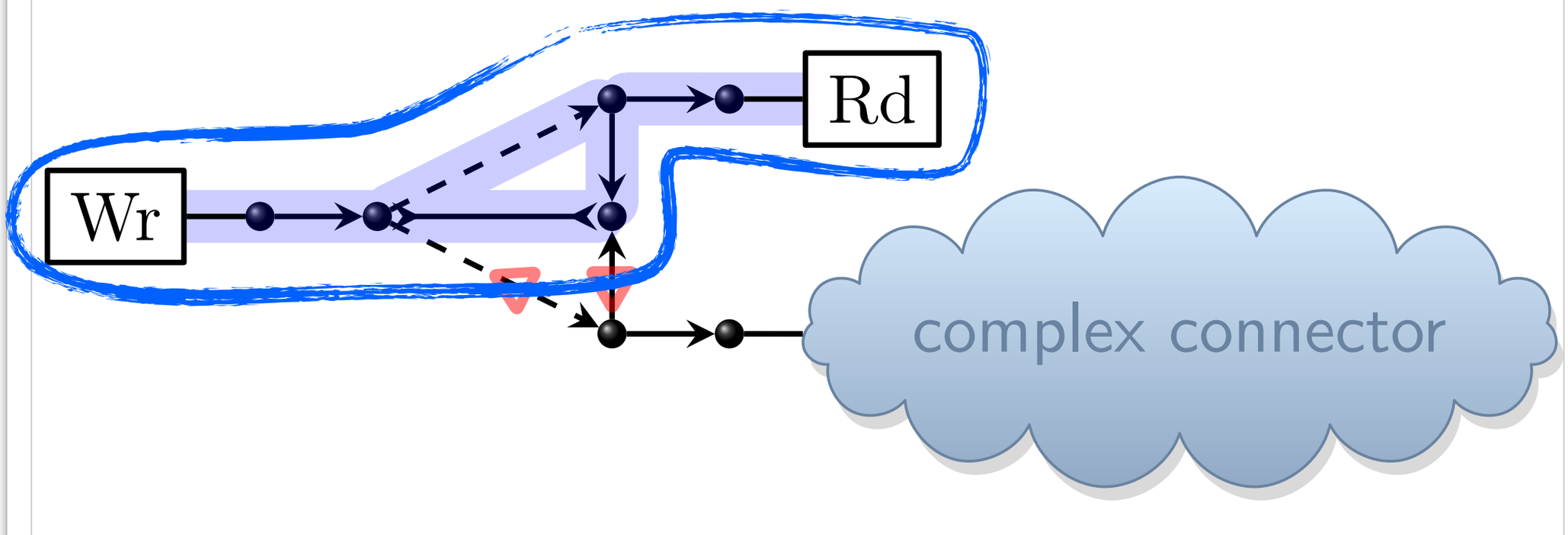
Connector colouring 3

- Compositional – composition operation is associative, commutative, and does not require post-processing.
- *Reasons* for the absence of flow are propagated.
- Expresses priority.
- 2 colours ⇔ constraint automata (without data)
- 3 colours: + expressive (⇔ intentional automata)

Outline

1. Context dependency
   - connector colouring 3 (CC3)[1]
2. Locality (concurrency)
   - partial connector colouring (PCC)[2]
3. Constraints
   - SAT solving with data for Reo[3]

[1] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency
[2] Dave Clarke and José Proença. Partial connector colouring
[3] Dave Clarke, José Proença, Alexander Lazovik, and Farhad Arbab, Channel-based coordination via constraint satisfaction
José Proença, Dave Clarke, Interactive interaction constraints



Locality (concurrency)

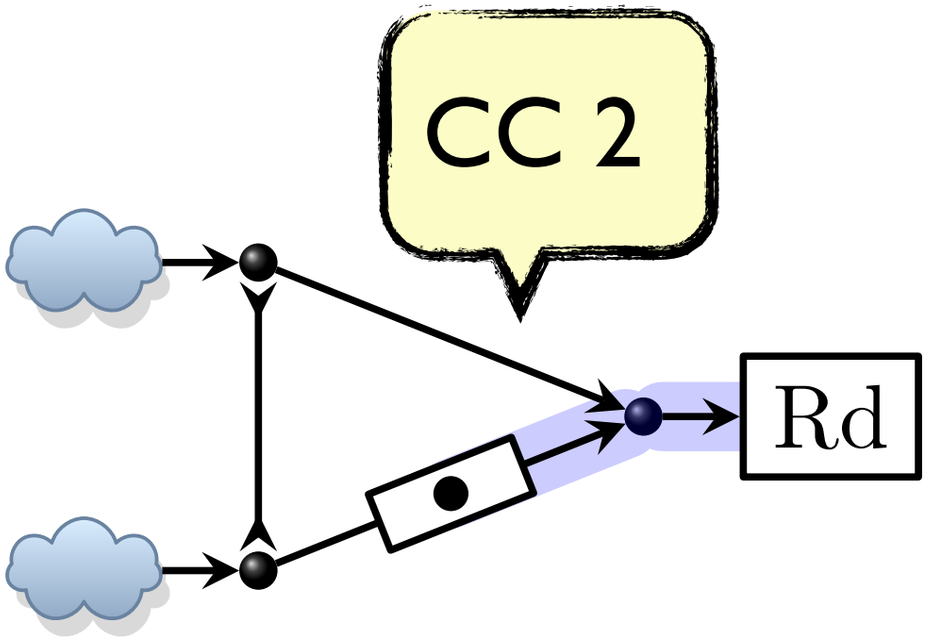
Motivation

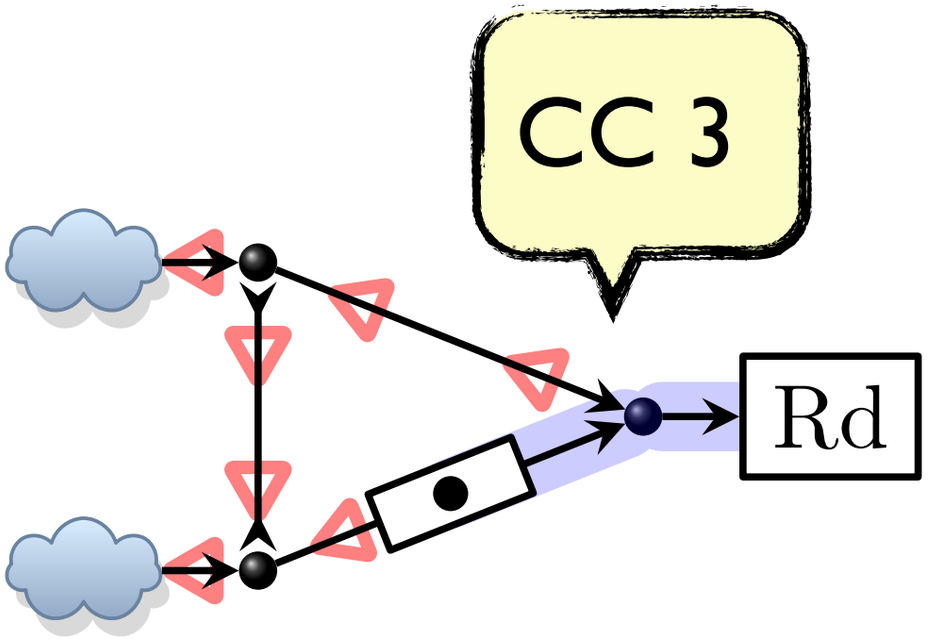
- Connector colouring is not optimal for distributed systems.
- All-or-nothing – all channels are needed to decide where data goes.
- Need to identify local flows that are not composed with the full connector.
- Model context dependency

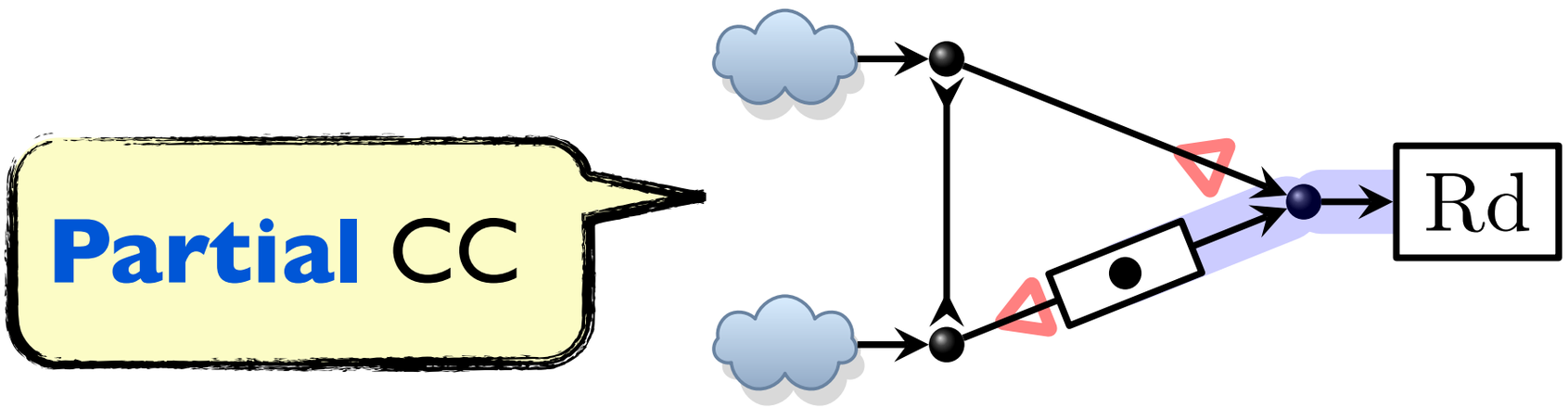
Problems

- 2 colours (or constraint automata):
  - assume primitives can make a *no-flow* step
- 3 colours:
  - cannot assume primitives have a no-flow colour – which direction would it be?
  - Idea?: add another no-flow colour, without direction, and assume all primitives have it...

Example





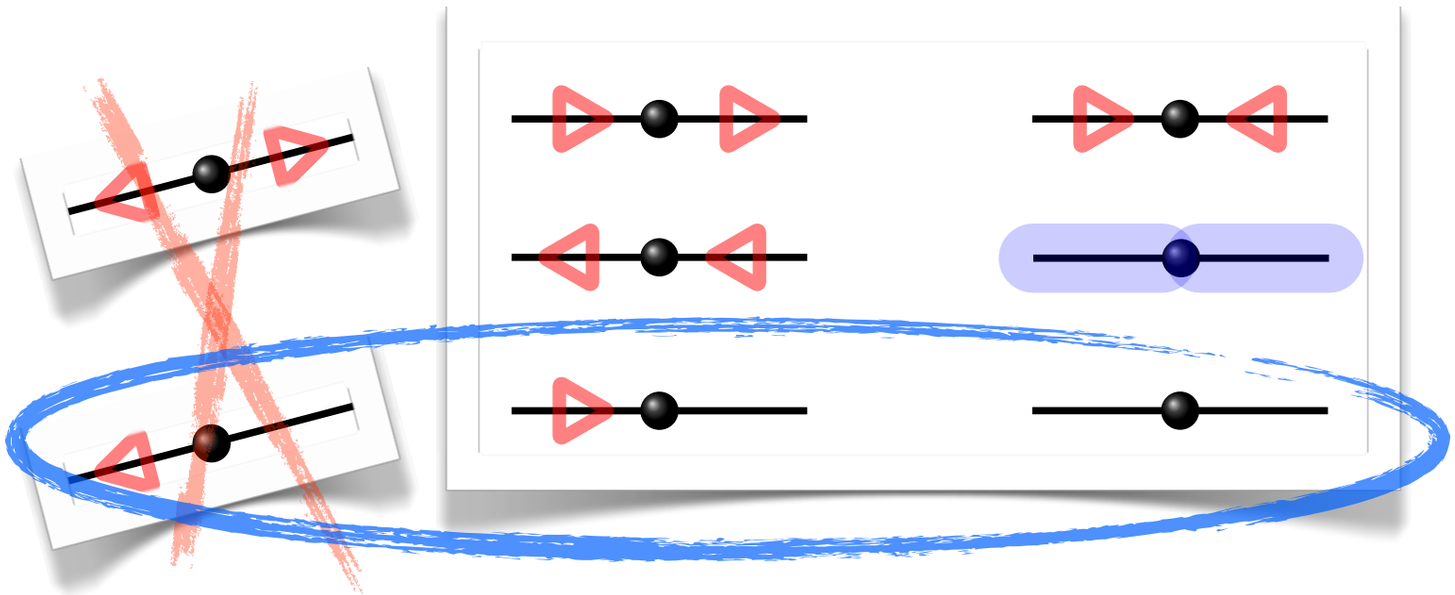


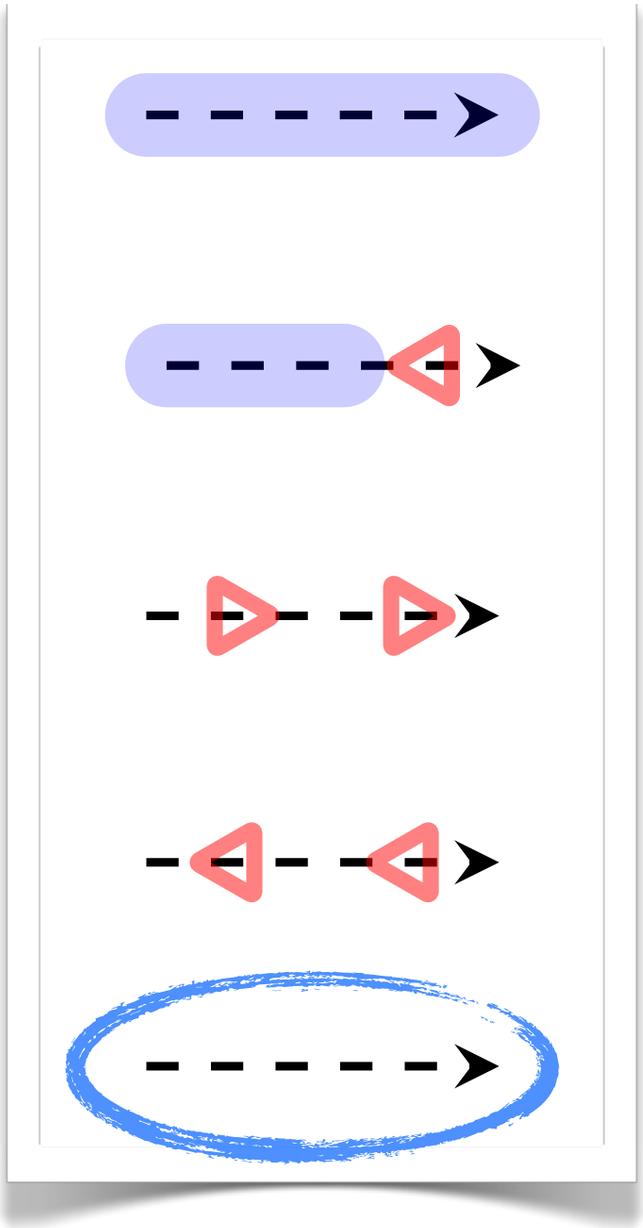
Partial connector colouring

- *Colouring*:

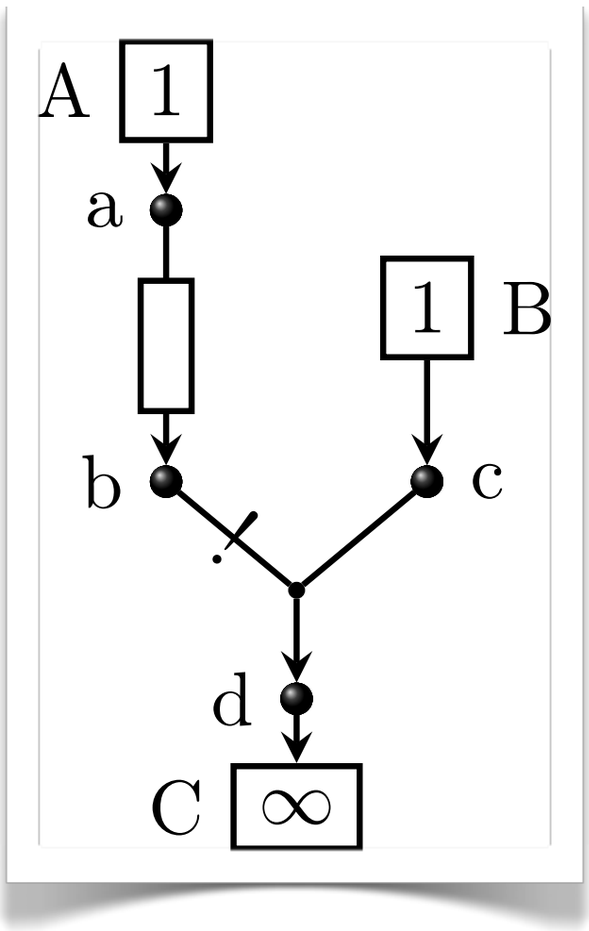
End → {Flow, GiveReason, GetReason}

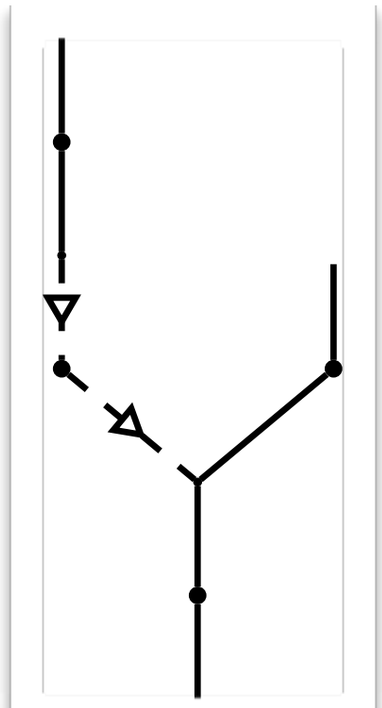
- *Composition* = matching colours:

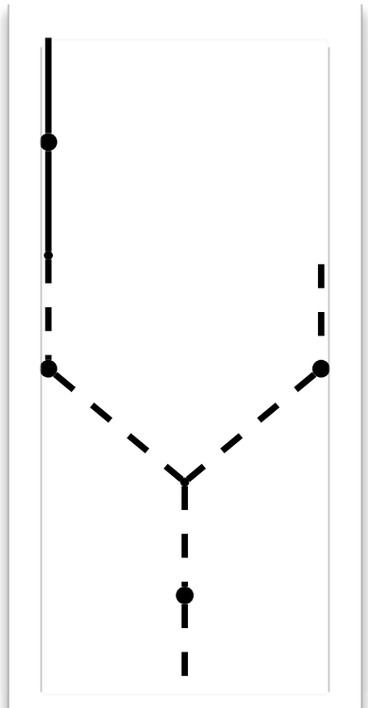


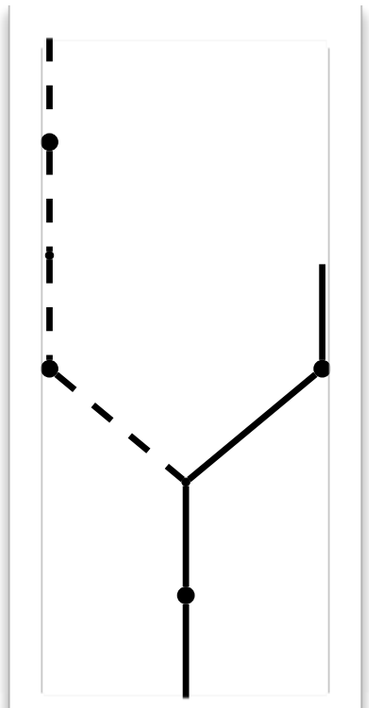


Partial CC vs. CC2/3

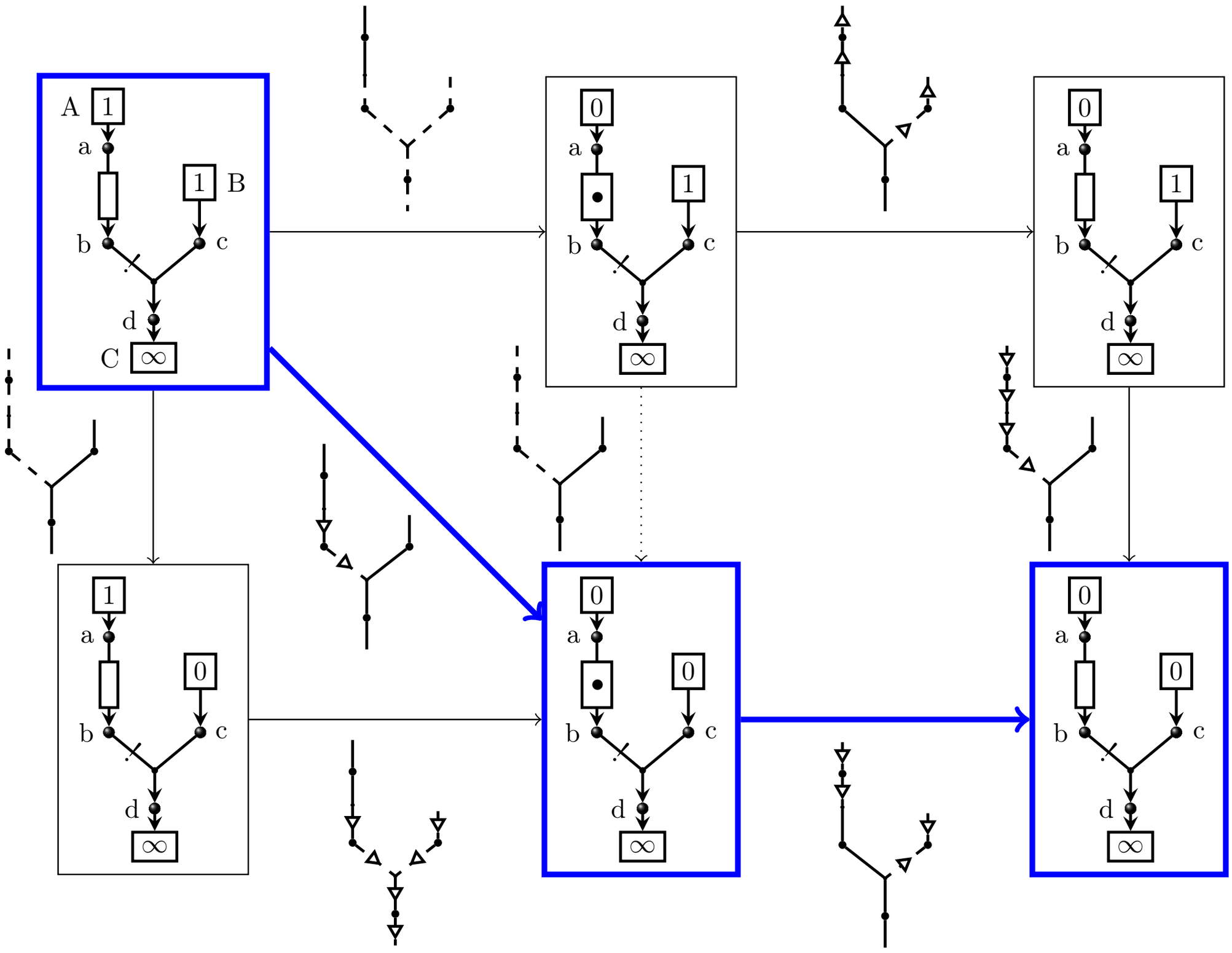




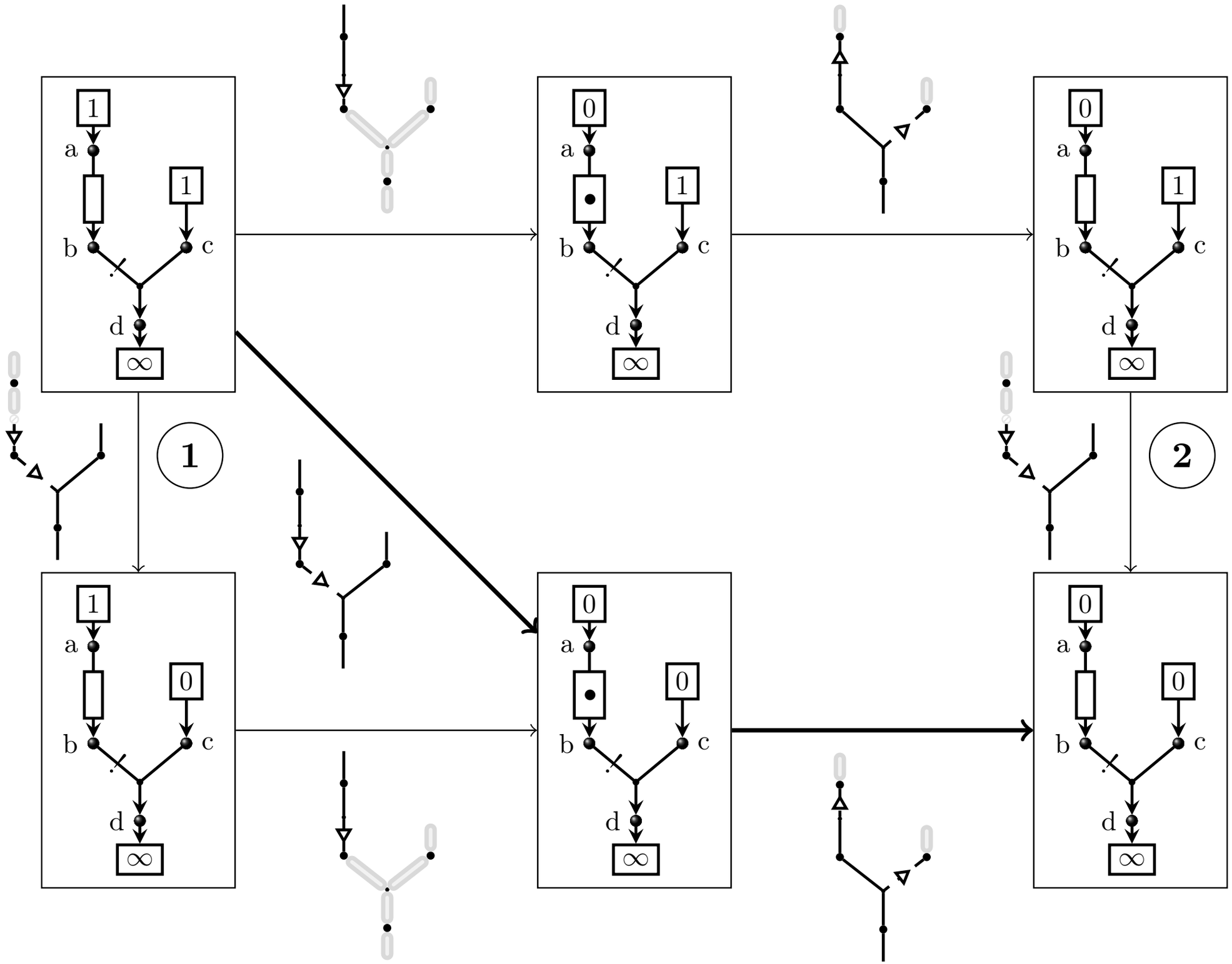




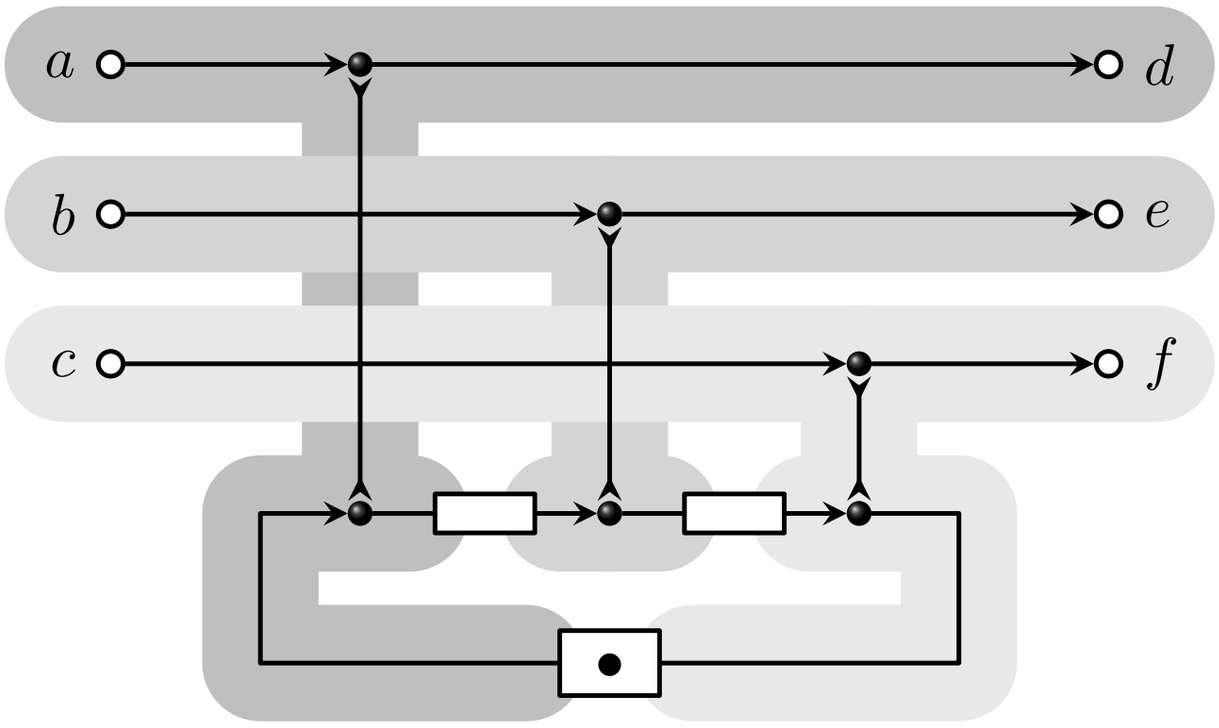
Partial CC vs. CC2/3



Partial CC vs. CC2/3

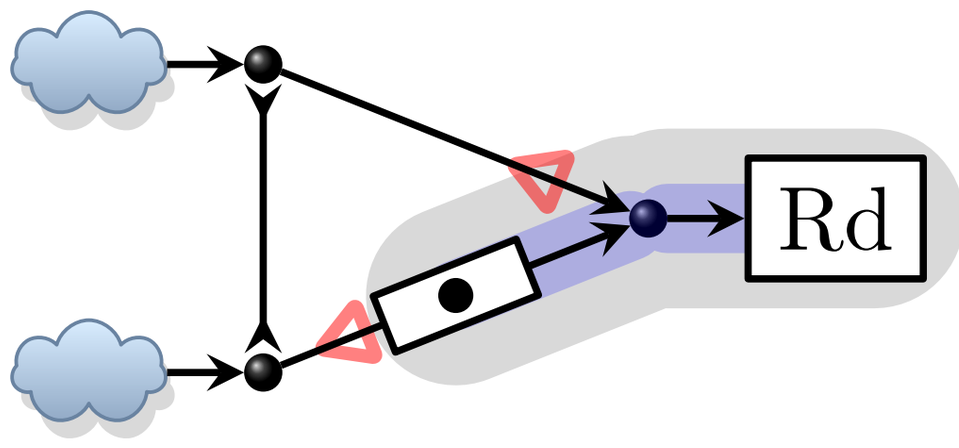


Synchronous regions



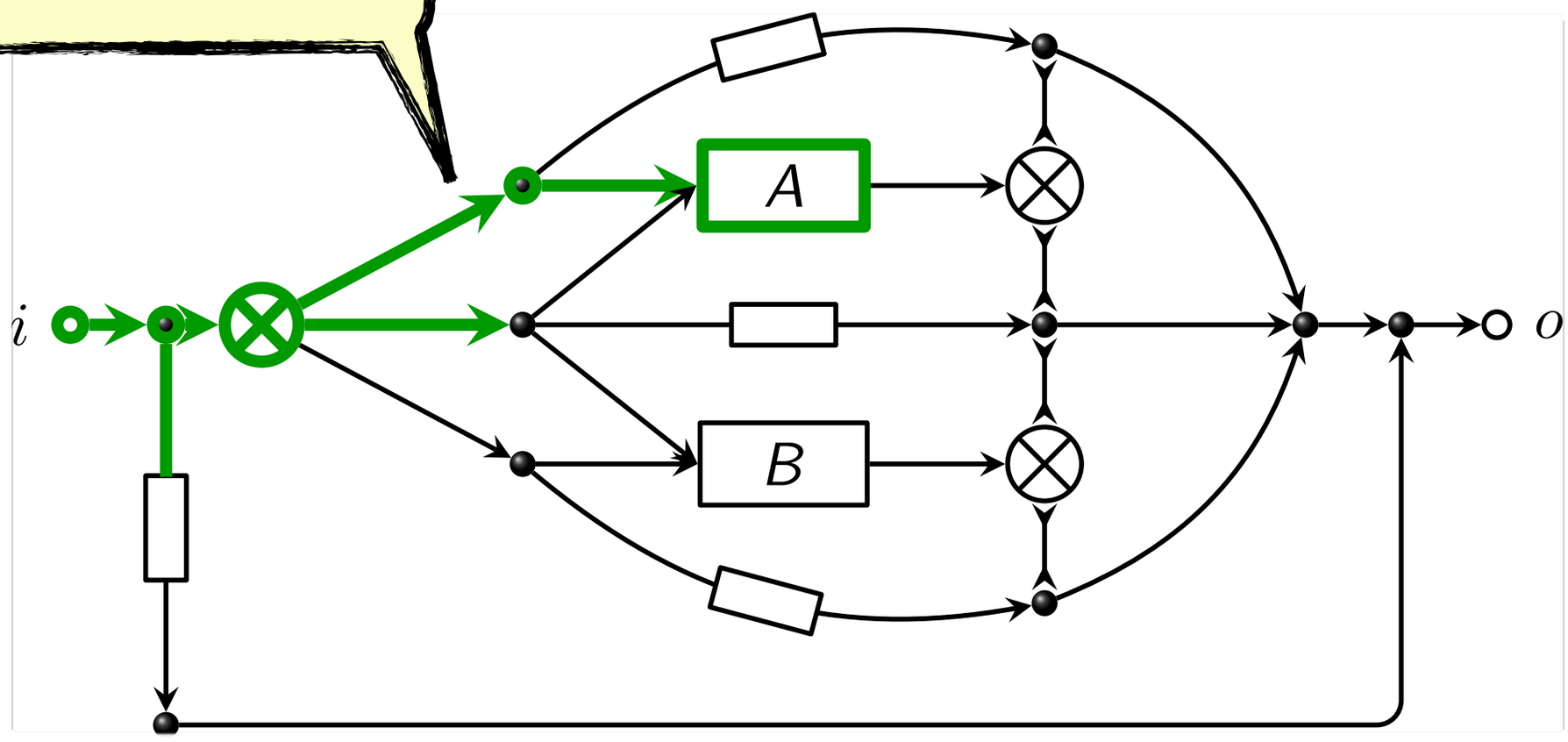
Static regions:
boundaries
=
FIFO's

Dynamic regions:
boundaries
=
GiveReason



In practice

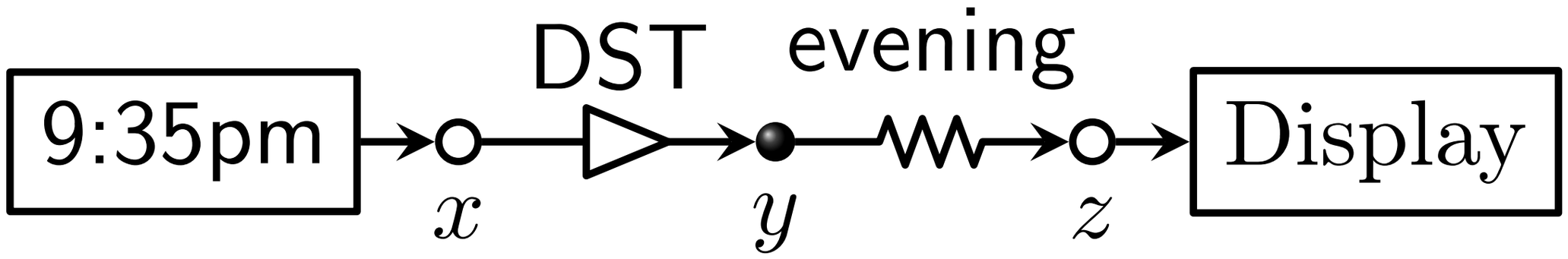
Shall I search now for a colouring?



Outline

1. Context dependency
   - connector colouring 3 (CC3)[1]
2. Locality (concurrency)
   - partial connector colouring (PCC)[2]
3. Constraints
   - SAT solving with data for Reo[3]

[1] Dave Clarke, David Costa, and Farhad Arbab. Connector colouring I: Synchronisation and context dependency
[2] Dave Clarke and José Proença. Partial connector colouring
[3] Dave Clarke, José Proença, Alexander Lazovik, and Farhad Arbab, Channel-based coordination via constraint satisfaction
José Proença, Dave Clarke, Interactive interaction constraints



$x \to \widehat{x} :=$ 9:35pm $\qquad x \leftrightarrow y \qquad y \to \widehat{y} :=$ DST$(\widehat{x})$

$(y \wedge$ evening$(\widehat{y})) \leftrightarrow z \qquad z \to \widehat{z} := \widehat{y}$

Reo as Constraints

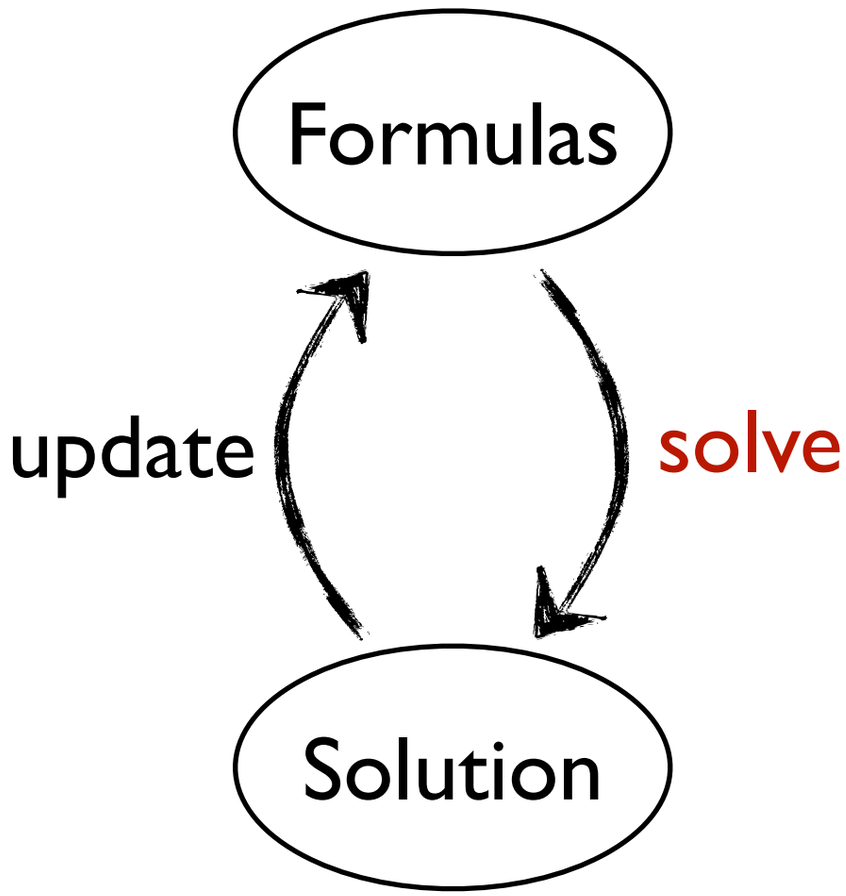
Implementation issues

Why another semantic model?

1. Backtracking and/or global arbitration are required
2. Some models are the basis for a centralized implementation (channels become redundant)
3. Some models cannot handle data aware behaviour
4. Channels take a significant role when resolving constraints (when distributed and acting as the only intermediaries)
5. No support for interaction with external primitives (e.g., data transformers/filters)

Coordination as constraint satisfaction

$a \longrightarrow b$	$a \leftrightarrow b$ $b \to \widehat{b} := \widehat{a}$
$a \dashrightarrow b$	$b \to a$ $b \to \widehat{b} := \widehat{a}$
$a \overset{P}{\rightsquigarrow} b$	$b \to \widehat{b} := \widehat{a}$ $(a \wedge P(\widehat{a})) \leftrightarrow b$
$a \overset{f}{\rightarrow\!\!\triangleright} b$	$a \leftrightarrow b$ $b \to \widehat{b} := f(\widehat{a})$

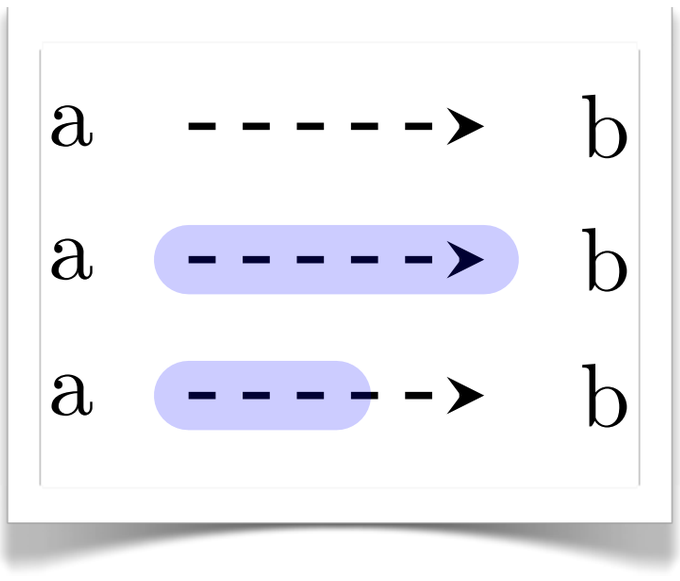


Building up constraints

- Connector colouring as constraints
- Data constraints
- Interactive constraints

CC2 as constraints

- [*Colouring*: End → {Flow, NoFlow}]
- *Formula:* Boolean over End
- *Composition* = conjunction

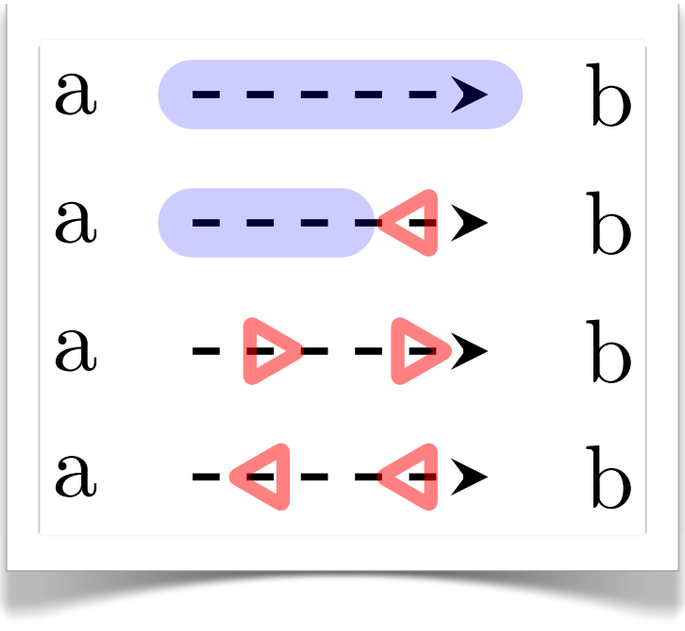


$$(\neg a \wedge \neg b) \quad \vee$$
$$(a \wedge b) \quad \vee$$
$$(a \wedge \neg b)$$

$$b \rightarrow a$$

CC3 as constraints

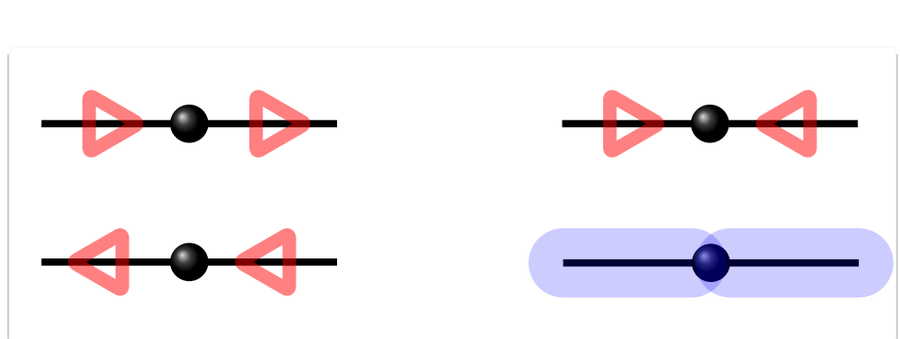
- [*Colouring*: End → {Flow, GiveReason, GetReason}]
- *Formula:* Boolean over End, End_{src}, End_{snk}
- a = flow on a ; a_{src} = give reason ; $\neg b_{snk}$ = get reason



$$b \rightarrow a \wedge$$
$$(a \wedge \neg b) \rightarrow (a \wedge \neg b_{snk}) \wedge$$
$$\neg a \rightarrow (\neg b \wedge \neg a_{src} \wedge b_{snk})$$

CC3 as constraints
Composition

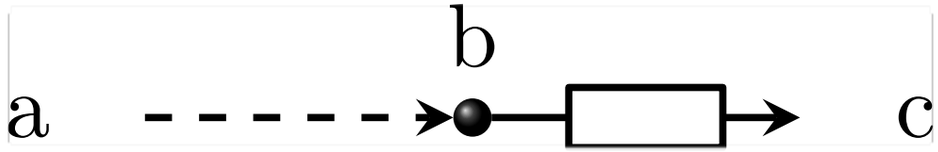
- a = flow on a ; a_{src} = give reason ; $\neg b_{snk}$ = get reason
- one-to-one composition: source to sink ends



$$\forall x \quad \cdot \quad x_{snk} \ \vee x_{src}$$

CC3 as constraints

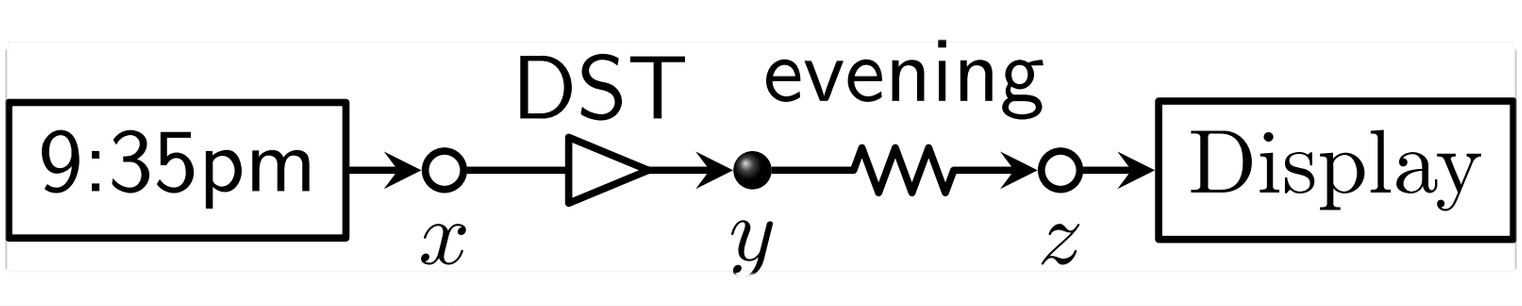
example



$$
\begin{aligned}
\phi \quad = \quad & b \rightarrow a \wedge \neg c \\
& (a \wedge \neg b) \rightarrow (a \wedge \neg b_{snk}) \wedge \\
& \neg a \rightarrow (\neg b \wedge \neg a_{src} \wedge b_{snk}) \wedge \\
& (\neg b \rightarrow \neg b_{src}) \wedge c_{snk} \wedge \\
& b_{src} \vee b_{snk}
\end{aligned}
$$

$$
\begin{aligned}
\{a \wedge b \wedge \neg c \wedge c_{snk}\} \quad & \models \quad \phi \\
\{\neg a \wedge \neg b \wedge \neg c \wedge \neg a_{src} \wedge b_{snk} \wedge \neg b_{src} \wedge c_{snk}\} \quad & \models \quad \phi
\end{aligned}
$$

Data constraints



$x \to \widehat{x} :=$ 9:35pm $\quad x \leftrightarrow y \quad y \to \widehat{y} :=$ DST$(\widehat{x})$

$(y \wedge$ evening$(\widehat{y})) \leftrightarrow z \quad z \to \widehat{z} := \widehat{y}$

How to solve this?

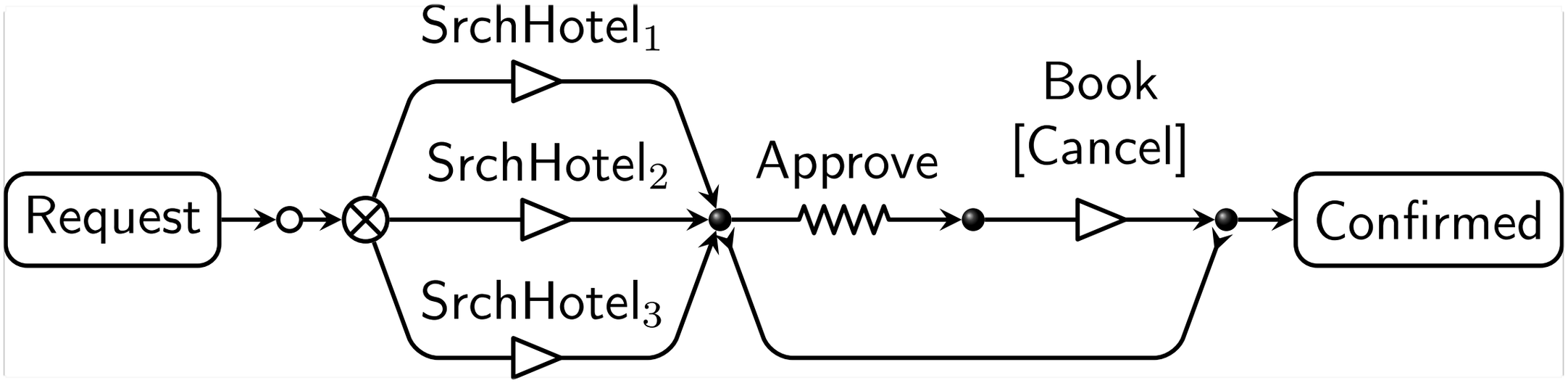
Solving data constraints

$$x \to \widehat{x} := \text{9:35pm} \qquad x \leftrightarrow y \qquad y \to \widehat{y} := \text{DST}(\widehat{x})$$
$$(y \wedge \text{evening}(\widehat{y})) \leftrightarrow z \qquad z \to \widehat{z} := \widehat{y}$$

- Use a SMT solver, e.g., over integers (Z3, Choco, Yices)
- Convert it into a boolean expression, then use a SAT solver (SAT4J, Z3, Choco)
  - better for complex data and interaction constraints

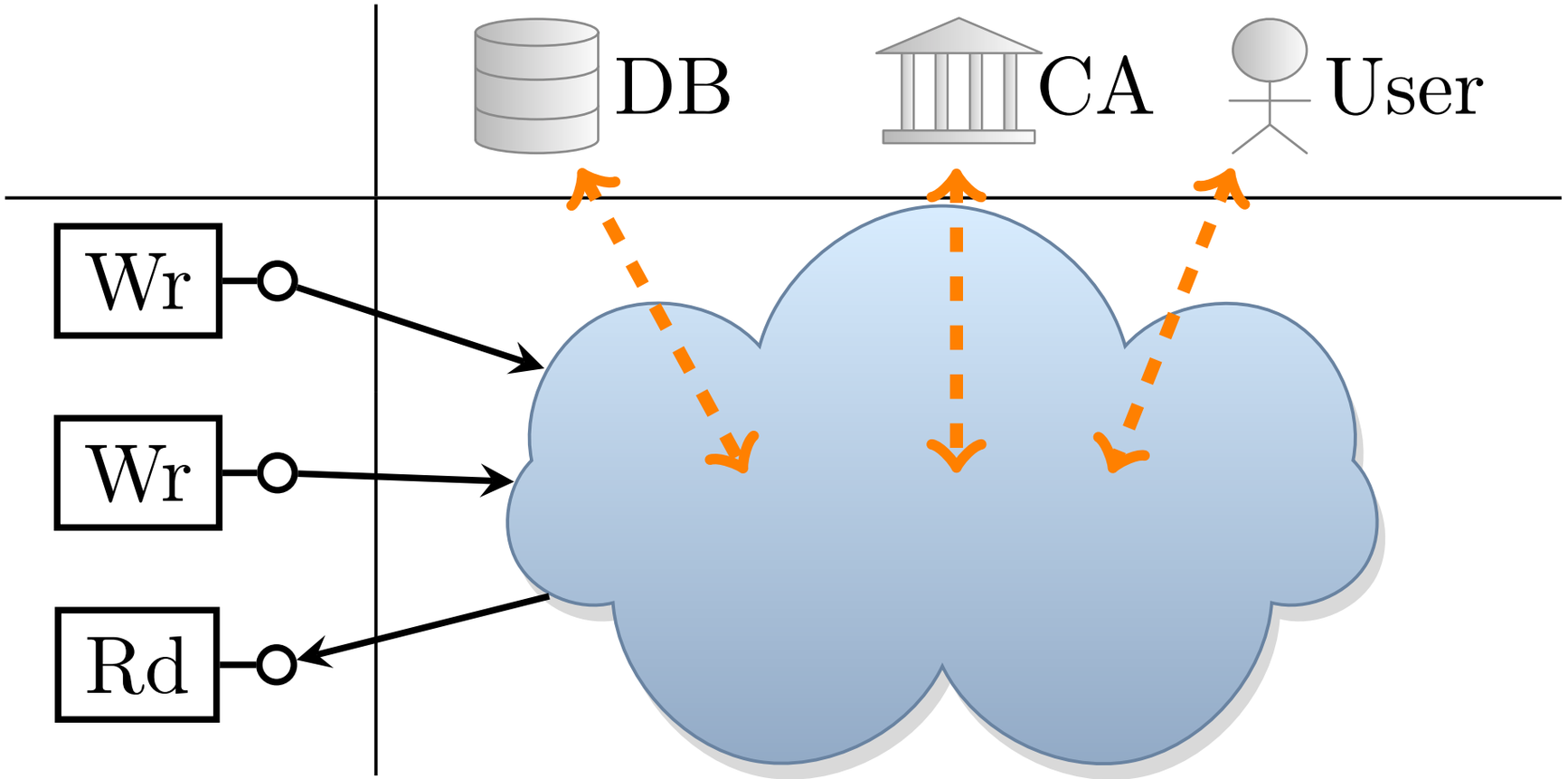
Interactive constraints

hotel booking



- Interaction with hotel repositories
- Interaction with users
- Interaction with hotels (availability & payment)

Interactive Interaction Constraints



Formulas

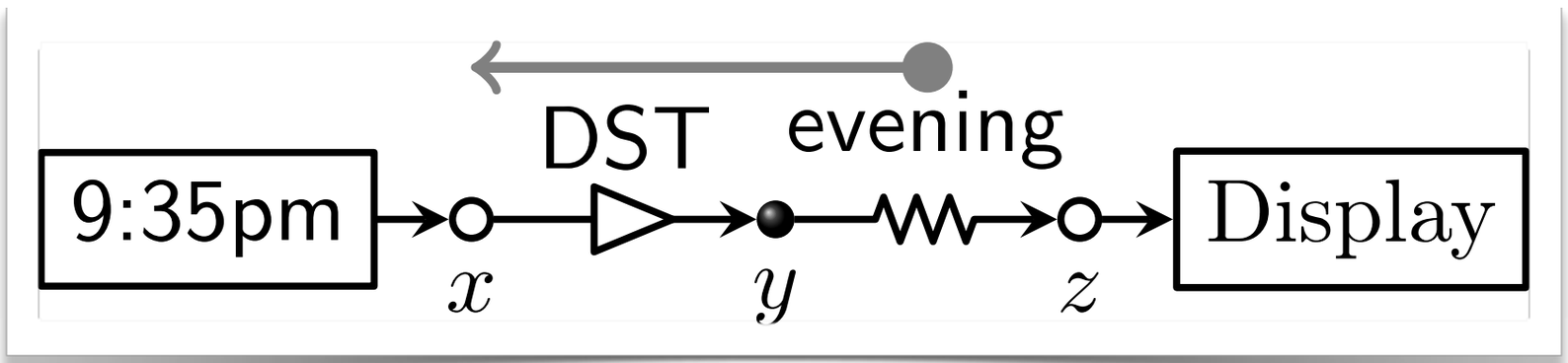
$$
\begin{array}{lll}
\psi & ::= & \phi \to s \mid \psi_1\ \psi_2 \mid \top & \text{(formulas)} \\
\phi & ::= & x \mid P(\widehat{x}) \mid \phi_1 \wedge \phi_2 \mid \neg\phi & \text{(guards)} \\
s & ::= & \phi \mid s_1 \wedge s_2 & \\
 & \mid & \widehat{x} := d \mid \widehat{x}_1 := \widehat{x}_2 \mid \widehat{x}_1 := f(\widehat{x}_2) & \text{(statements)}
\end{array}
$$

Well-defined formula:
no data-loops; single assignment; data source

Predicate abstraction

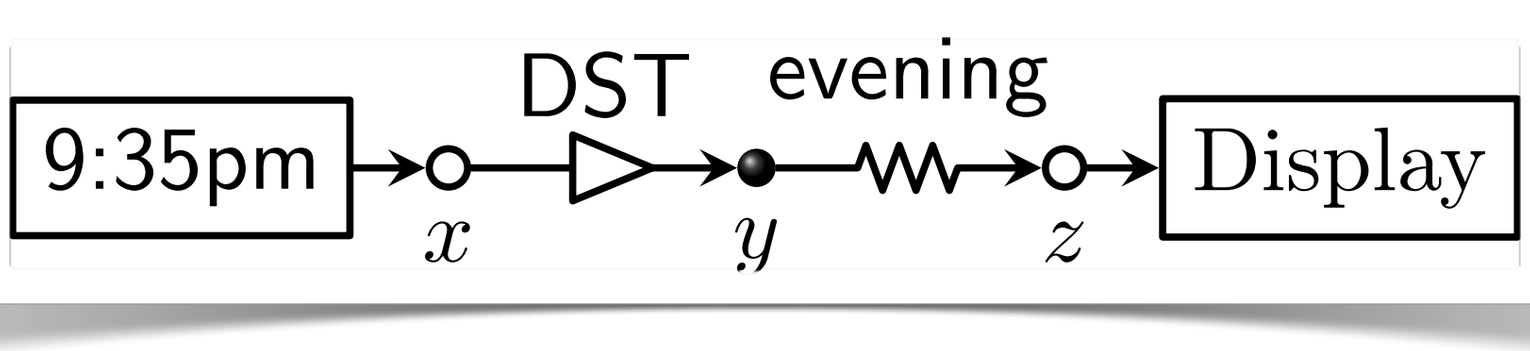
- Find data dependencies of predicates
- Replace data variables by boolean variables that guide the predicates

Predicate abstraction



$$\mathcal{D}_x = \{\mathsf{ev.dst}\}$$
$$\mathcal{D}_y = \{\mathsf{ev}\}$$
$$\mathcal{D}_z = \emptyset$$

Predicate abstraction



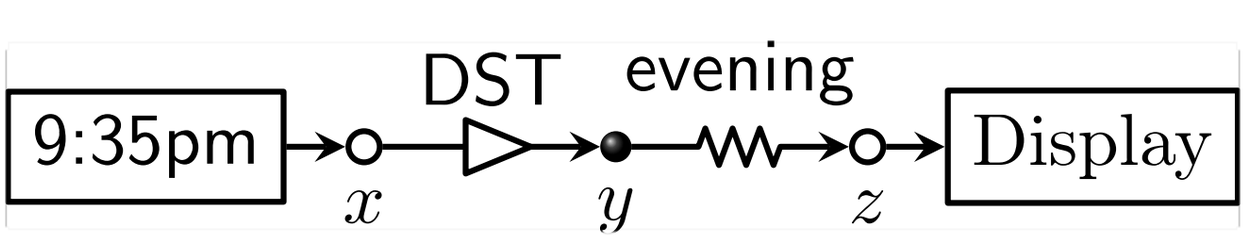
original

$$x \to \widehat{x} := \text{9:35pm} \qquad x \leftrightarrow y \qquad y \to \widehat{y} := \text{DST}(\widehat{x})$$
$$(y \wedge \text{evening}(\widehat{y})) \leftrightarrow z \qquad z \to \widehat{z} := \widehat{y}$$

boolean

$$x \to \widehat{x}_{\text{ev.dst}} := [\text{evening}(\text{DST}(\text{9:35pm}))] \qquad x \leftrightarrow y$$
$$y \to \widehat{y}_{\text{ev}} := \widehat{x}_{\text{ev.dst}} \qquad (y \wedge \widehat{y}_{\text{ev}}) \leftrightarrow z$$

Interaction with Choco



boolean

$$x \to \widehat{x}_{\text{ev.dst}} := [\text{evening}(\text{DST}(9{:}35\text{pm}))] \qquad x \leftrightarrow y$$

$$y \to \widehat{y}_{\text{ev}} := \widehat{x}_{\text{ev.dst}} \qquad (y \wedge \widehat{y}_{\text{ev}}) \leftrightarrow z$$

interactive

$$x \to \text{XPred}(\text{ev.dst}, x, 9{:}35\text{pm}) \qquad x \leftrightarrow y$$

$$y \to \widehat{y}_{\text{ev}} := \widehat{x}_{\text{ev.dst}} \qquad (y \wedge \widehat{y}_{\text{ev}}) \leftrightarrow z$$

Interaction with Choco

- instance of a Choco constraint
- reacts when x or $\widehat{x}_{\text{ev.dst}}$ is instantiated
- $\neg x \Rightarrow \widehat{x}_{\text{ev.dst}}$ can be anything
- $x \Rightarrow \widehat{x}_{\text{ev.dst}} = \text{ev}(\text{dst}(9{:}35\text{pm}))$

9:35pm

boolean

$x \to \widehat{x}_{\text{ev.dst}} = [\text{evening}(\text{DST}(9{:}35\text{pm}))] \qquad x \leftrightarrow y$

$y \to \widehat{y}_{\text{ev}} := \widehat{x}_{\text{ev.dst}} \qquad (y \wedge \widehat{y}_{\text{ev}}) \leftrightarrow z$

interactive

$x \to \text{XPred}(\text{ev.dst}, x, 9{:}35\text{pm}) \qquad x \leftrightarrow y$

$y \to \widehat{y}_{\text{ev}} := \widehat{x}_{\text{ev.dst}} \qquad (y \wedge \widehat{y}_{\text{ev}}) \leftrightarrow z$

Avoiding pre-processing

~~SAT solver~~ → SMT solver (integers)

original

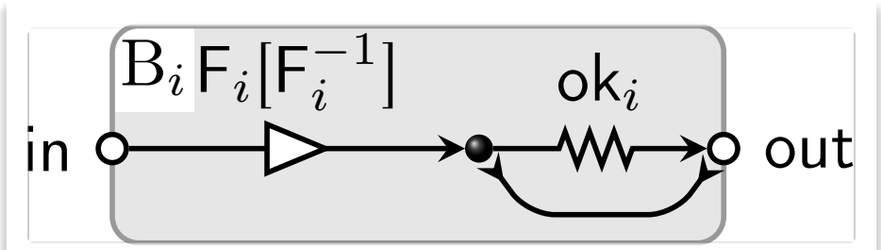
$$x \to \widehat{x} := \text{9:35pm} \qquad x \leftrightarrow y \qquad y \to \widehat{y} := \text{DST}(\widehat{x})$$
$$(y \wedge \text{evening}(\widehat{y})) \leftrightarrow z \qquad z \to \widehat{z} := \widehat{y}$$

integer

$$x \to \widehat{x} := 0 \qquad x \leftrightarrow y \qquad y \to \widehat{y} := \text{DST}^{int}(\widehat{x})$$
$$(y \wedge \text{evening}^{int}(\widehat{y})) \leftrightarrow z \qquad z \to \widehat{z} := \widehat{y}$$

```
0 -> 9:35pm
...
```

Scala / Java implementation

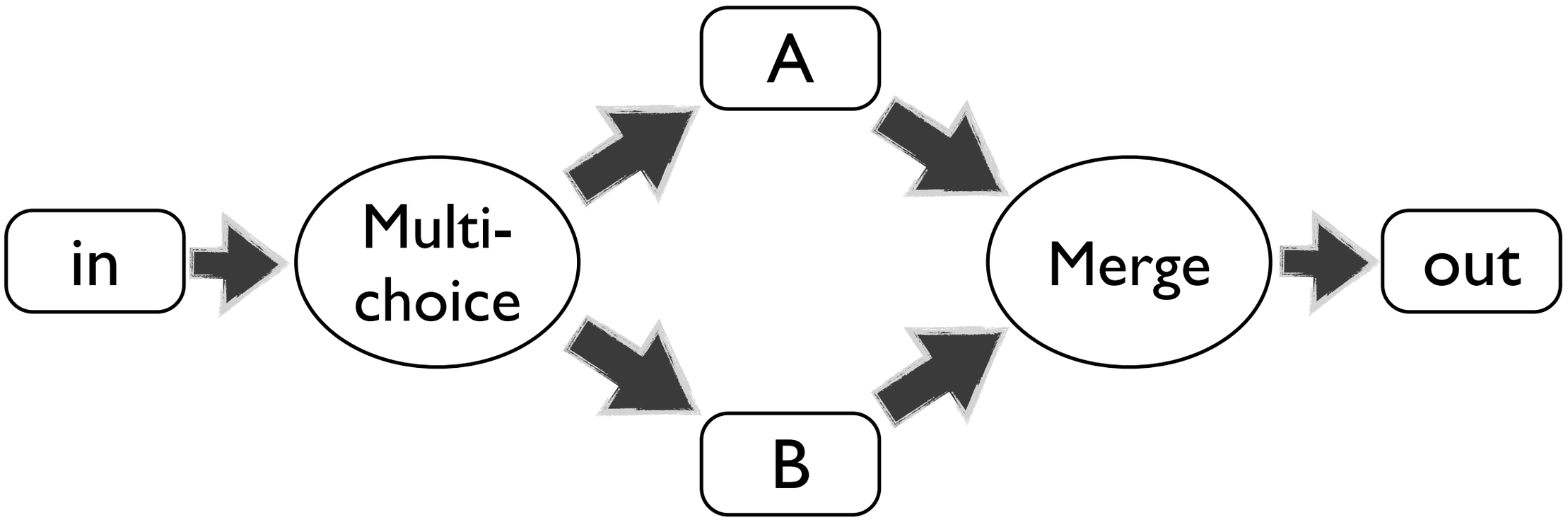



```
val f = Function("f") {
  case s: String => /* do something */
}

val finv = Function("f^-1") {
  case s: String => /* do something */
}

val ok = Predicate("ok") {
  case s: String => /* do something */
}

val connector =
  writer("in",List("a","b")) ++
  transf("in","x",f,finv)    ++
  filter("x","out",ok)       ++
  reader("out",2)

connector.run()
```

```
class Filter(as: String, bs: String,
             uid: Int, g: Guard)
             extends ... {

  val a = Var(flowVar(as,uid))
  val b = Var(flowVar(bs,uid))

  def getConstraints = Formula(
    b --> a,
    b --> (b := a),
    b --> g,
    (a /\ g) --> b
  )

  /* override def update(s:
       Option[Solution]) = ... */
}
```

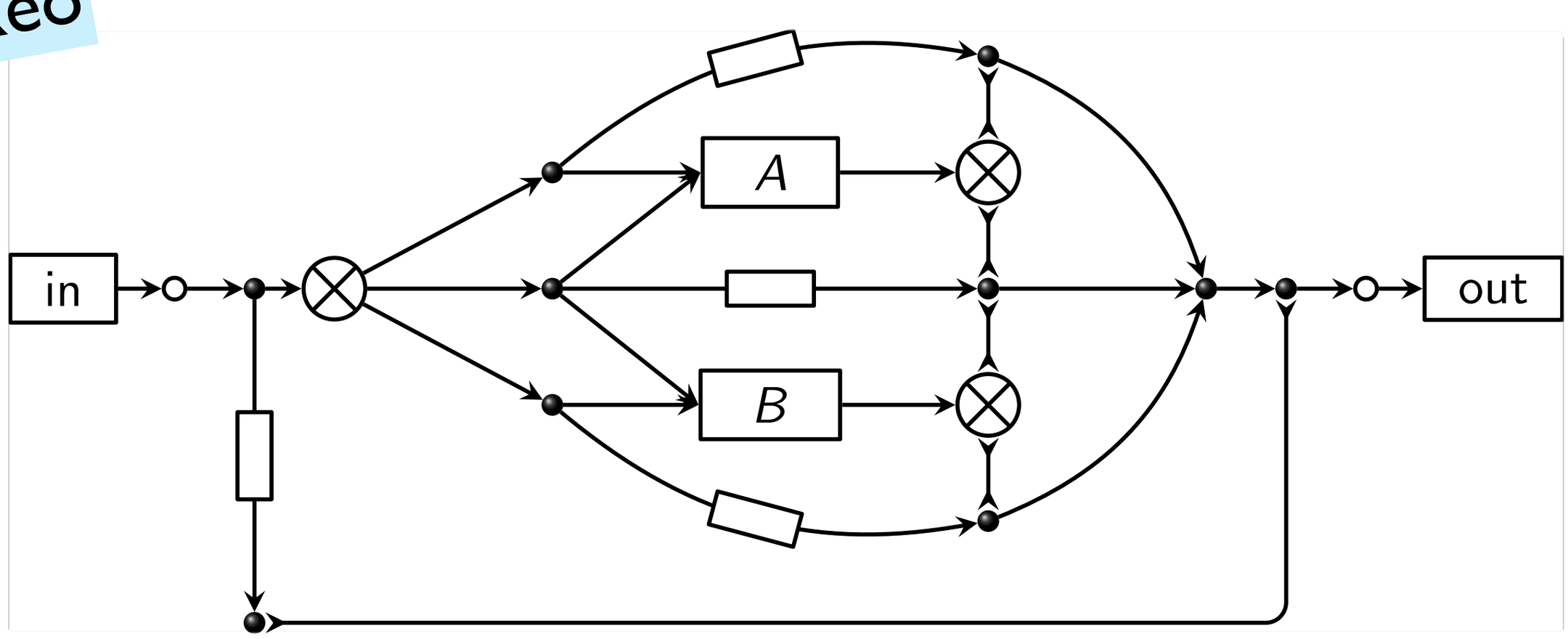
Extra slides

Synchronous coordination



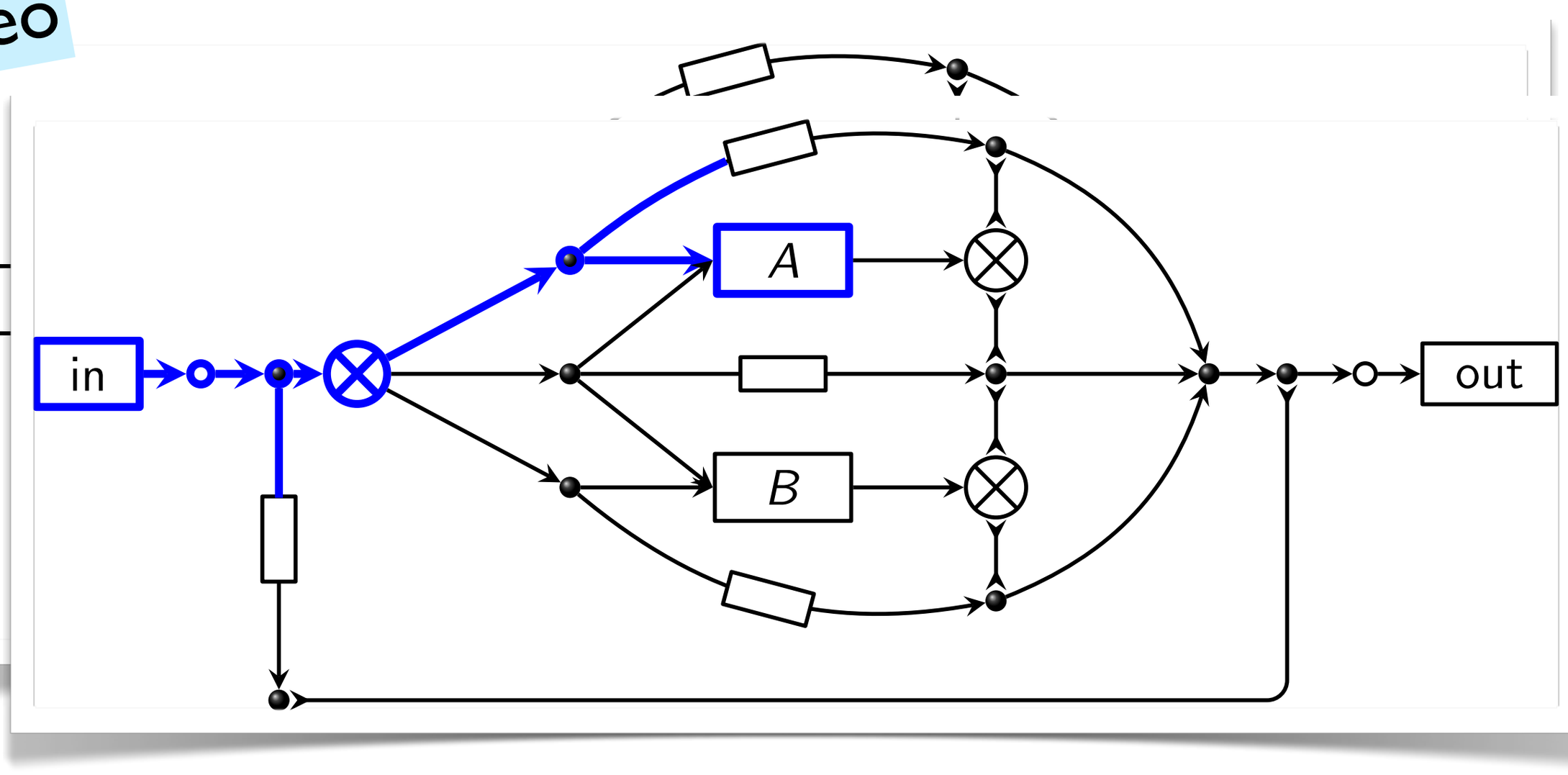
Synchronous coordination

Reo



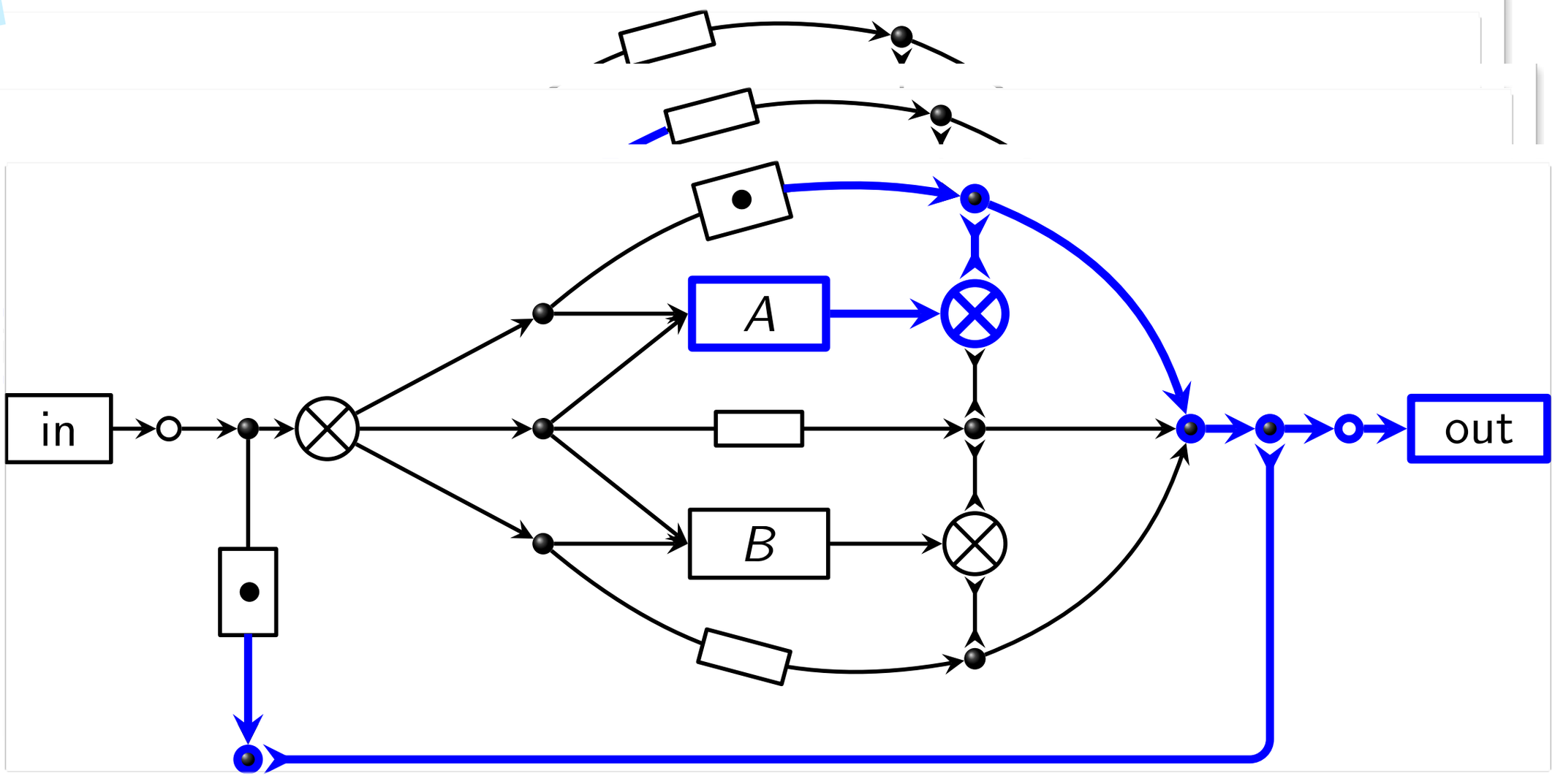
Synchronous coordination

Reo

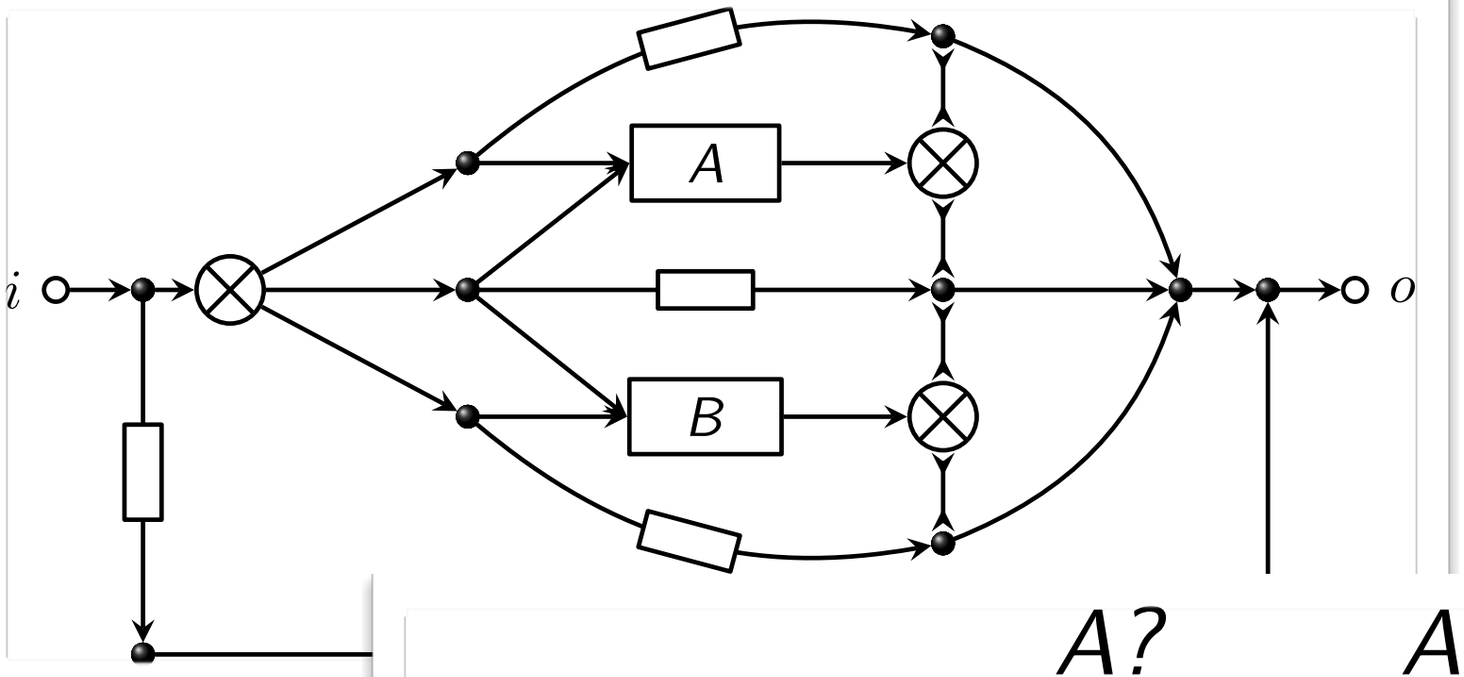


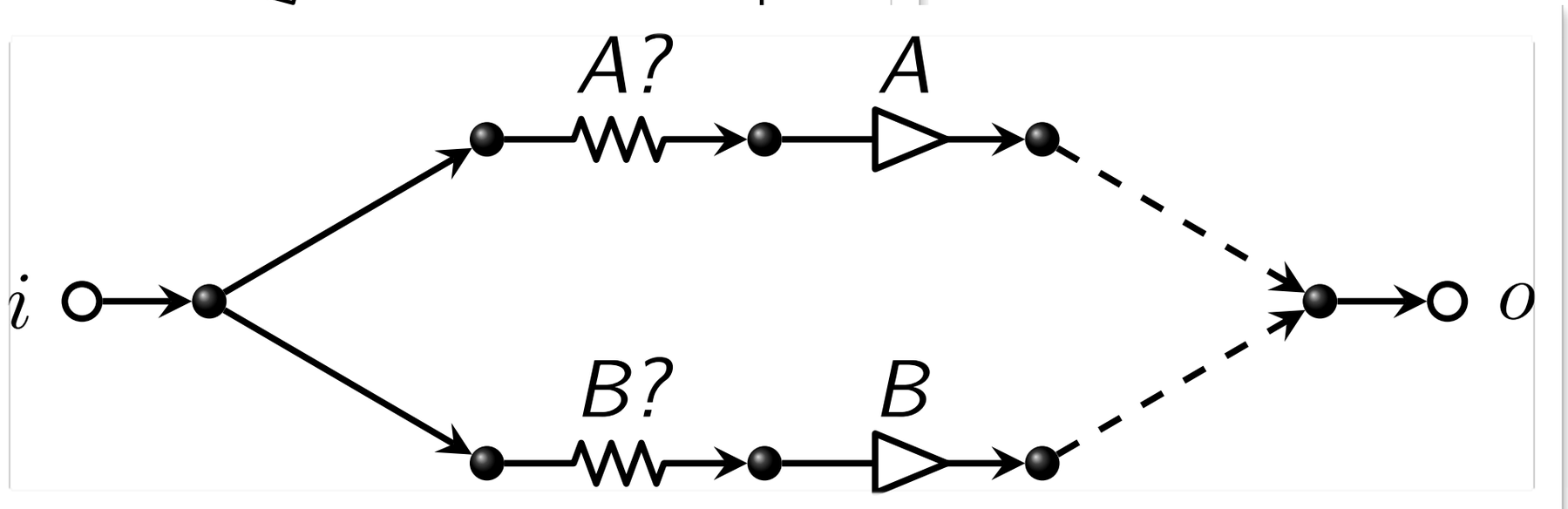
Synchronous coordination

Reo

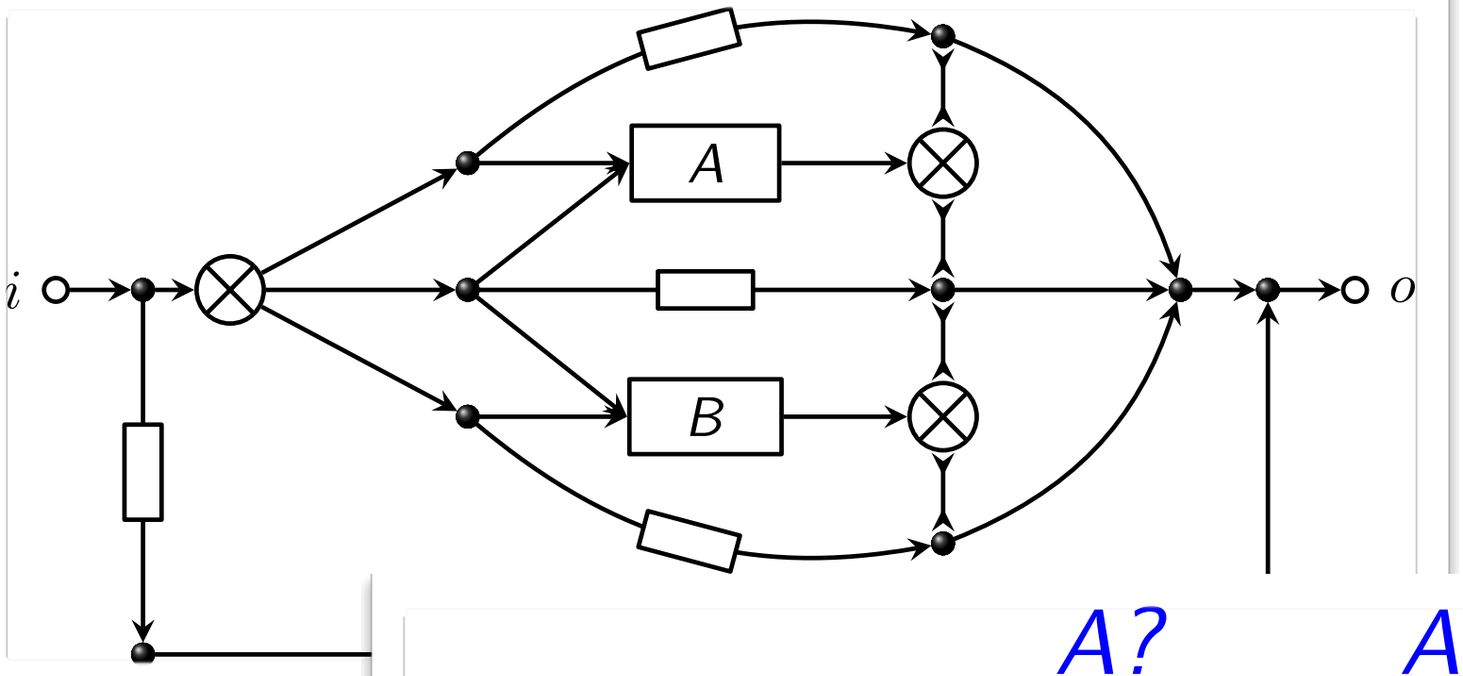


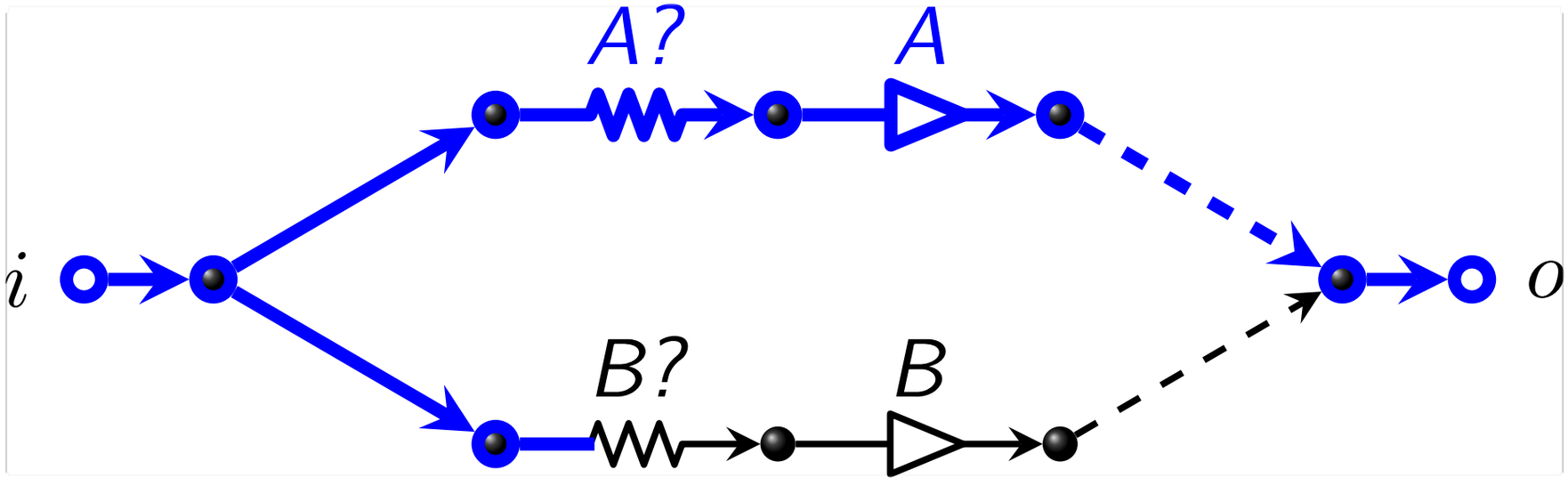
More synchronous





More synchronous





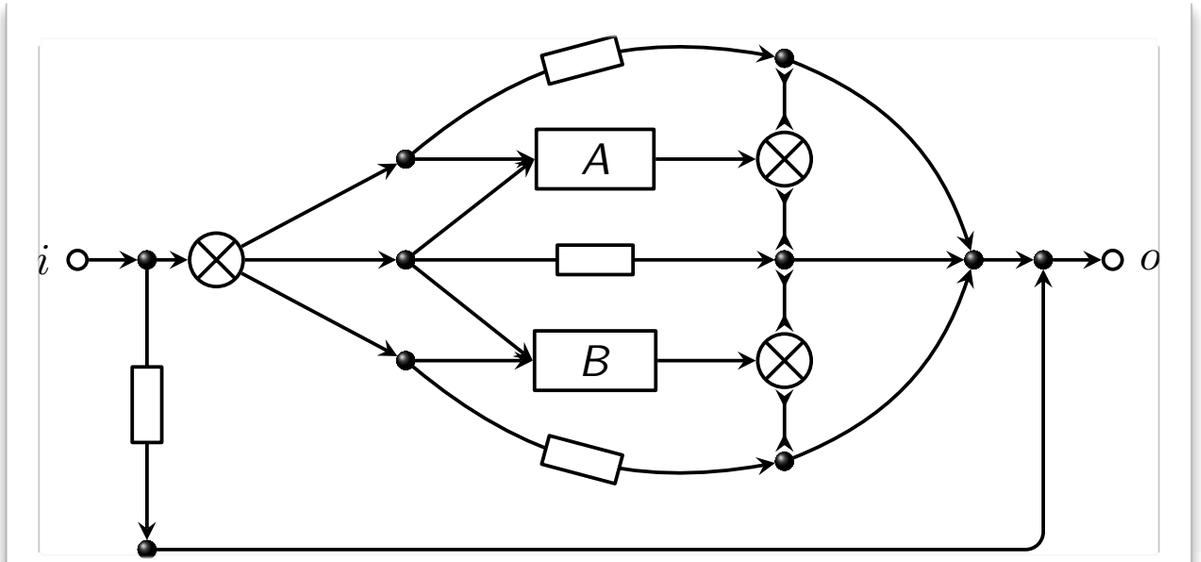
Solving constraints

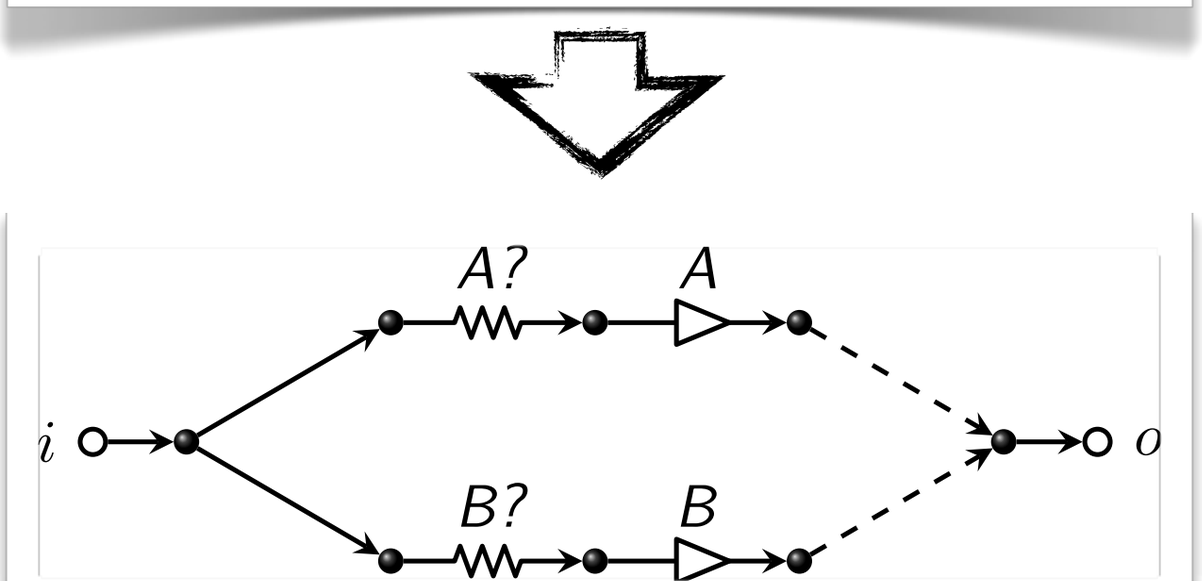
- Formulas
- Data formula ⇨ Boolean formula
- Interaction with the Choco solver
- Scala (Java) prototype implementation

Wrapping up

- Interactive constraint solving

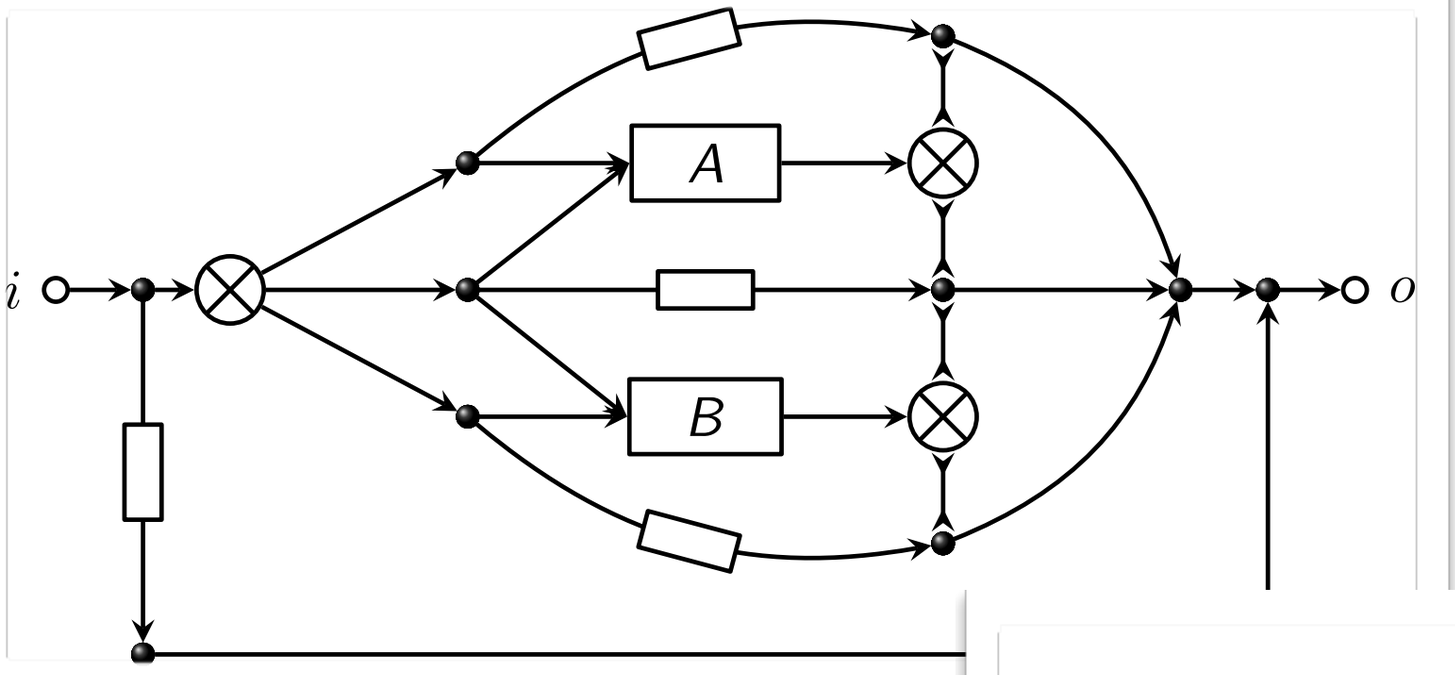
- Expose the atomicity of Reo to components

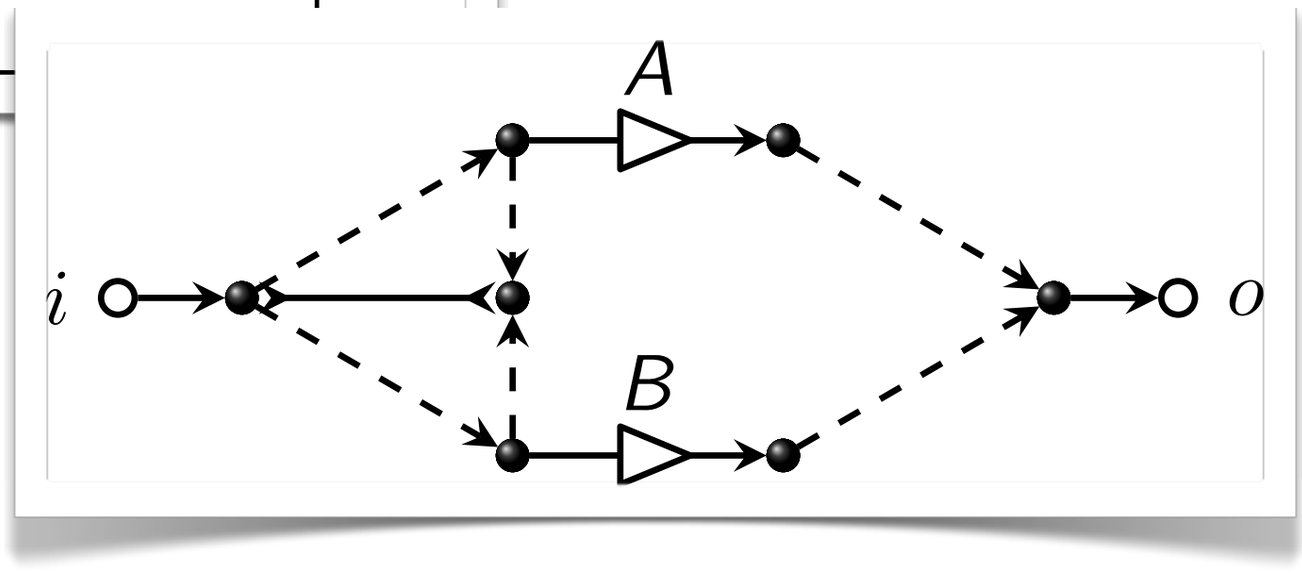




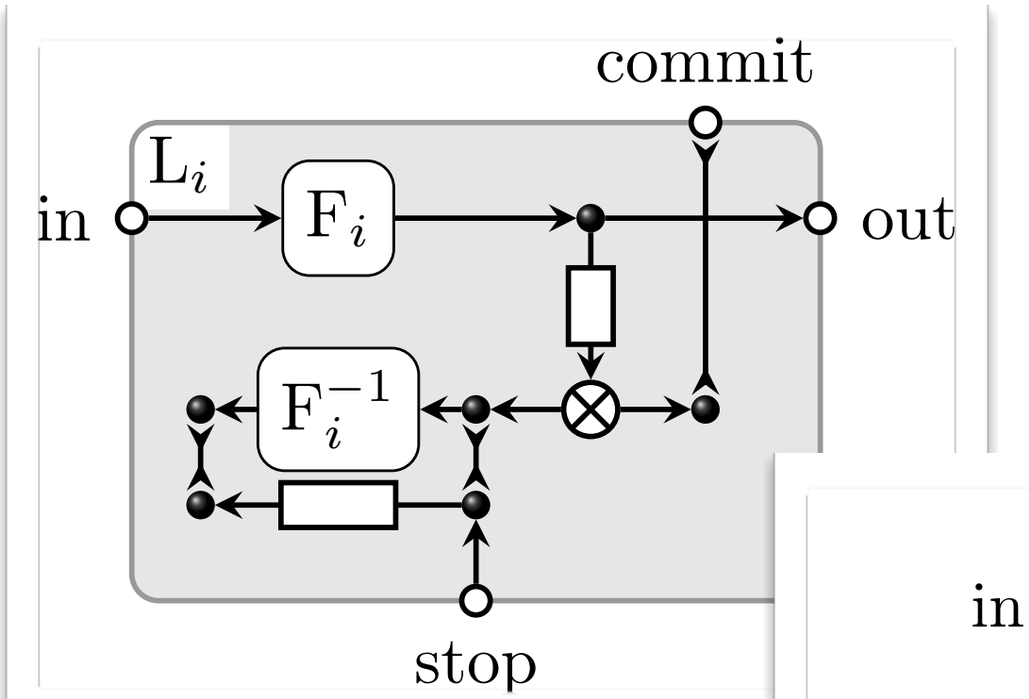
- Beyond this paper: Avoiding pre-processing using an SMT solver

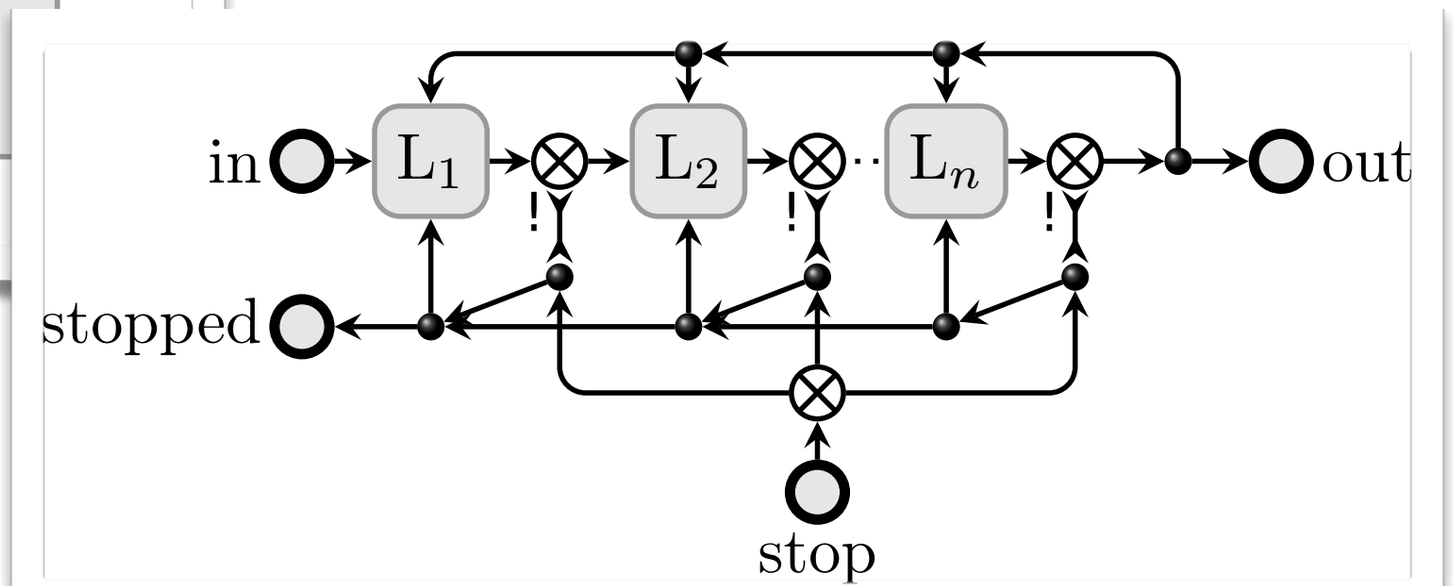
Interactive Interaction Constraints



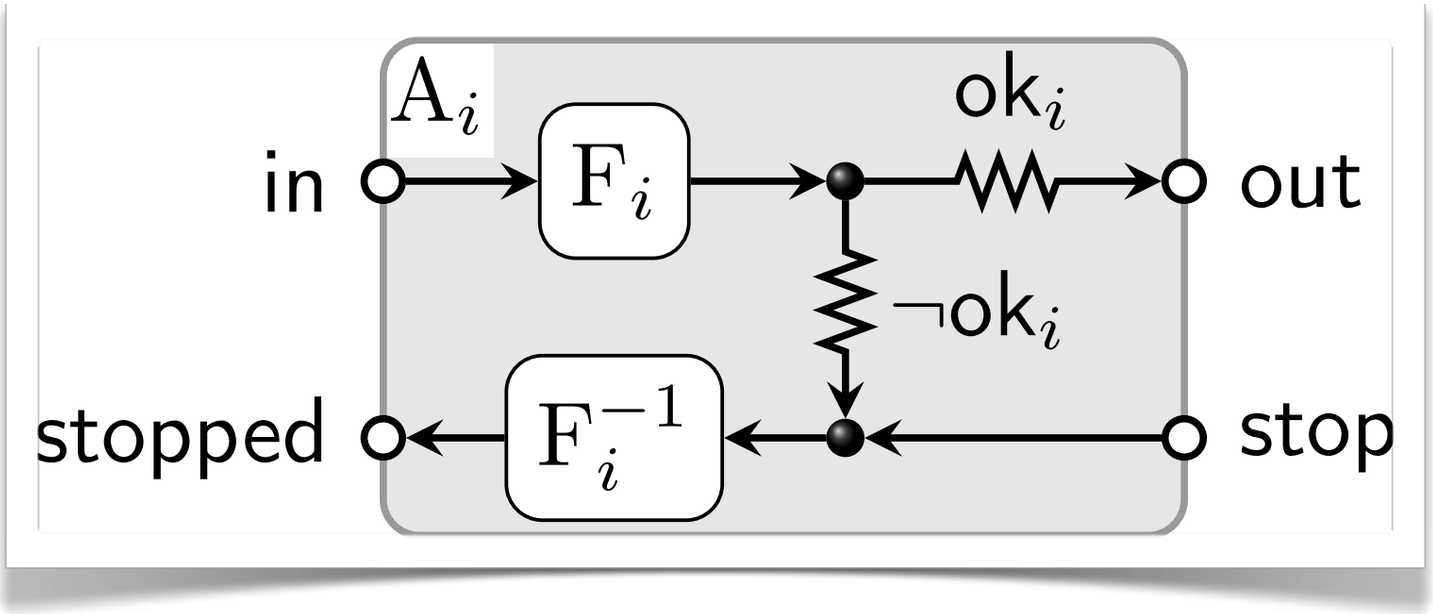


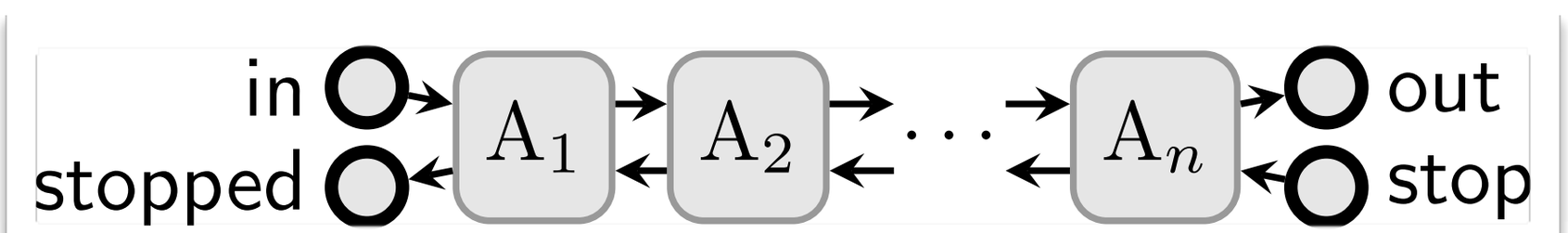
Long-running transactions



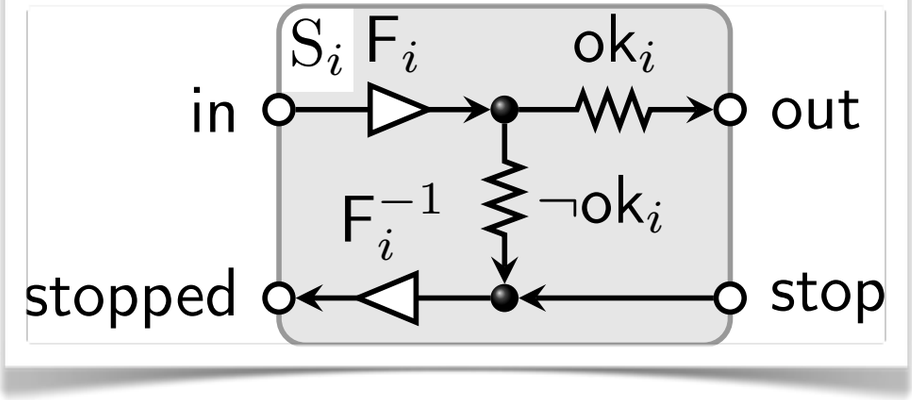


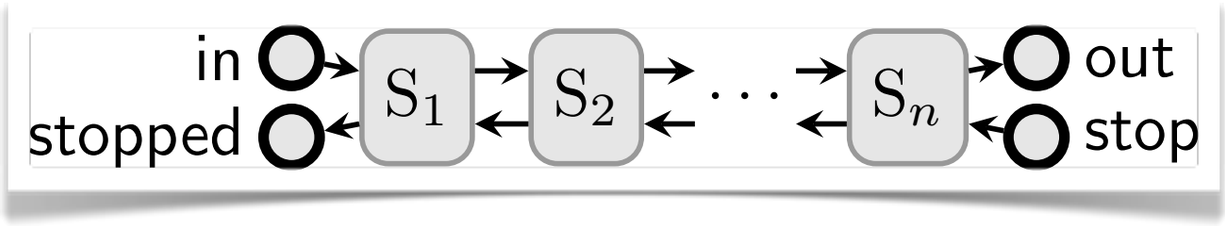
Asynchronous transactions

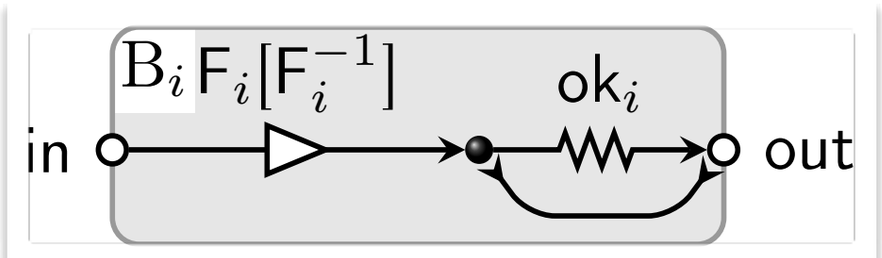


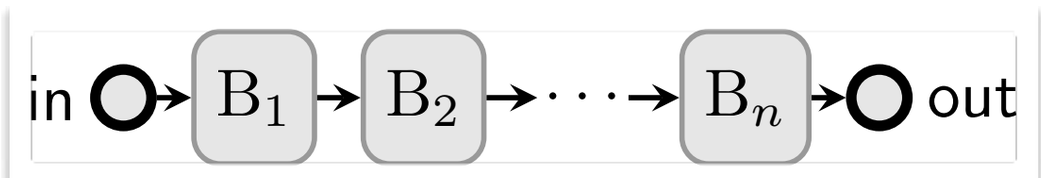


Synchronous transactions

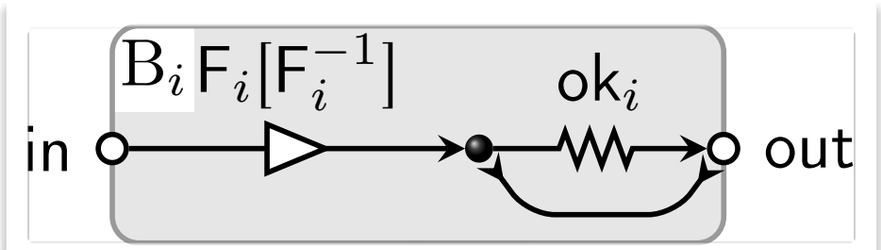








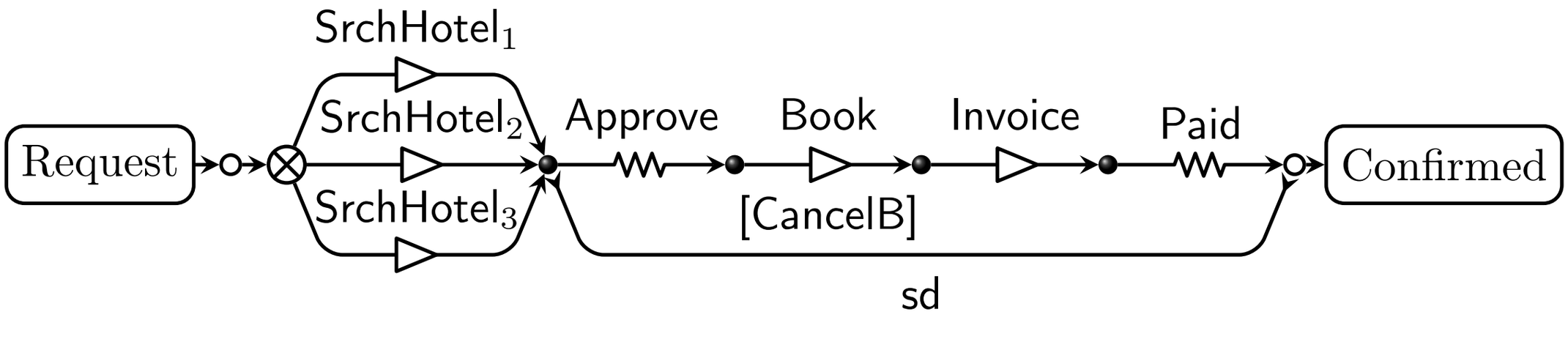
Scala / Java implementation




```
val f = Function("f") {
  case s: String => /* do something */
}

val finv = Function("f^-1") {
  case s: String => /* do something */
}

val ok = Predicate("ok") {
  case s: String => /* do something */
}

val connector =
  writer("in",List("a","b")) ++
  transf("in","x",f,finv)    ++
  filter("x","out",ok)       ++
  reader("out",2)

connector.run()
```

```
class Filter(as: String, bs: String,
             uid: Int, g: Guard)
             extends ... {

  val a = Var(flowVar(as,uid))
  val b = Var(flowVar(bs,uid))

  def getConstraints = Formula(
    b --> a,
    b --> (b := a),
    b --> g,
    (a /\ g) --> b
  )

  /* override def update(s:
       Option[Solution]) = ... */
}
```

Hotel booking



- Interaction with users
- Interaction with hotel repositories
- Interaction with hotels (availability)
- Interaction with hotels (payment)

```
object HotelReservation extends App {

  case class Req(val content:String)

  def srchHotel(i:Int) =
        Function("SearchHotel-"+i){
    case r:Req ⇒ i match {
      case 1 ⇒ List("F1","Ibis","Mercury")
      case 2 ⇒ List("B&B","YHostel")
      case _ ⇒ List("HotelA","HotelB")
    }
  }
  val approve = Predicate("approve"){
    case l:List[String] ⇒
      println("approve: "+l.mkString(",
            ")+". [y,n]")
      readChar() == 'y'
  }
  val book = Function("book"){
    case l : List[String] ⇒
      println("Options: "+l.mkString(", ")+
        ". Which one? (1.."+l.length+")")
      val res = readInt()
      l(res-1)
  }
  val cancelB = Function("cancelB"){
    case x ⇒ println("canceling "+x+".")
  }
  val invoice = Function("invoice"){
    case x ⇒ println("invoice for "+x+".")
  }
```

```
  val pay = Predicate("paid"){
    case x ⇒ if (x == "Ibis") {
      println("paid for Ibis")
      true
    }
    else {
      println("not paid for "+x)
      false
    }
  }



  // Connector definition
  val connector =
    writer("req",List(Req("req1"),
          Req("req2"))) ++
    nexrouter("req",List("h1","h2","h3")) ++
    transf("h1","h1o",srchHotel(1)) ++
    transf("h2","h2o",srchHotel(2)) ++
    transf("h3","h3o",srchHotel(3)) ++
    nmerger(List("h1o","h2o","h3o"),"hs") ++
    filter("hs","ap",approve) ++
    sdrain("hs","ap") ++
    transf("ap","bk",book,cancelB) ++
    monitor("bk","inv",invoice) ++
    filter("inv","paid",paid) ++
    reader("paid",5)

  connector.run()
}
```